INF 215 Software Testing Report

Apache Tika

Dinghang Yu 76004312
Ke Xu 15962797
Yi Tan 11767614

Work Distribution

Task	Pages	Dinghang Yu ID: 76004312	Ke Xu ID: 15962797	Yi Tan ID: 11767614
Building the system	3~7	10	10	10
Preliminary testing and coverage	7~9	3	3	3
new BDD scenarios and new JUnit test cases in BDD style (4 multiples * 5 points)	10	20		
new BDD scenarios and new JUnit test cases in BDD style (4 multiples * 5 points)	12		20	
new BDD scenarios and new JUnit test cases in BDD style (4 multiples * 5 points)	11			20
Apply a static analyzer (PMD) and document both a justifiable and a false positive warning	12~16	15		
Apply a static analyzer (FindBugs) and document both a justifiable and a false positive warning	16~19		15	
Apply a static analyzer (Checkstyle) and document both a justifiable and a false positive warning	19~23			15
Apply a reverse-engineering tool to reveal the code structure by generating UML Class Diagrams, Call Graphs (2 multiples * 15 points)	26~32	30		
Apply a reverse-engineering tool to reveal the code structure by generating Call Graphs, Control Flow Graphs (2 multiples * 15 points)	31~34		30	
Apply a reverse-engineering tool to reveal the code structure by generating UML Class Diagrams, Control Flow Graphs(2 multiples * 15 points)	26~30, 32~34			30
Find a documented and verified bug report that has no automated test case that reproduces it, and create such an automated test case. (2 multiples * 20 points)	35~41	40		
Find a documented and verified bug report that has no automated test case that reproduces it, and create such an automated test case. (2 multiples * 20 points)	47~52		40	
Find a documented and verified bug report that has no automated test case that reproduces it, and create such an automated test case. (2 multiples * 20 points)	41~46			40
Total points per member:		**118**	**118**	**118**

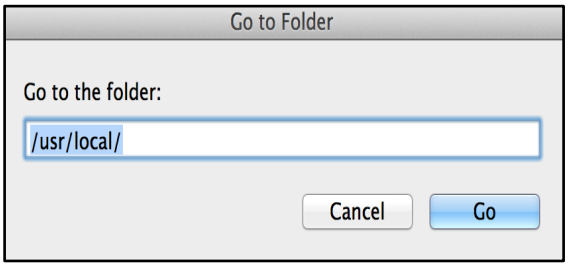
1. Building the System

1.1 Download tika: http://www.apache.org/dyn/closer.cgi/tika/tika-1.5-src.zip, and uncompress it;

1.2 Since tika-1.5 is a Maven project, so we need to install Maven;

1.2.1 Download Maven: http://maven.apache.org/download.cgi - Installation, I choose the "apache-maven-3.2.1-bin.zip" and uncompress it;

1.2.2 Go to folder "/usr/local/";

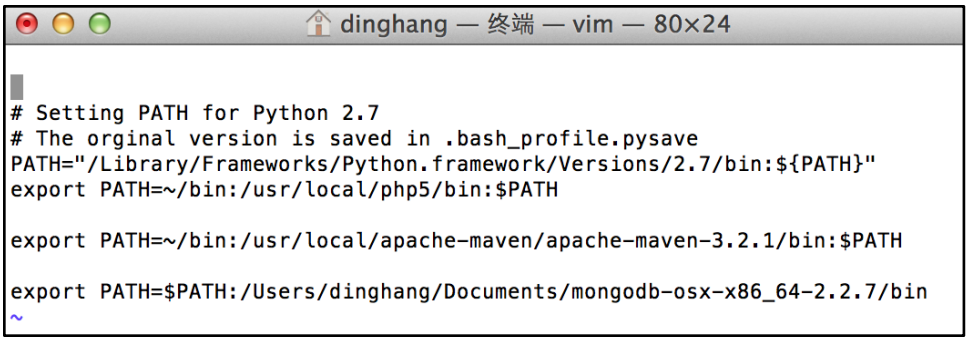


1.2.3 In that folder, create a new folder called "apache-maven", then, move your uncompressed Maven "apache-maven-3.2.1" into that folder;

1.2.4 Use terminal to add Maven environment variable for quick use in the future;

1.2.5 Open a new terminal, type in "~/.bash_profile" to open the setting file;

1.2.6 Type "i" to start insert. Add a line "PATH=~/bin:/usr/local/apache-maven/apache-maven-3.2.1/bin:$PATH" as the picture shows. Then press "esc" button followed by ":". After that, type in "wq" and press enter to quit;

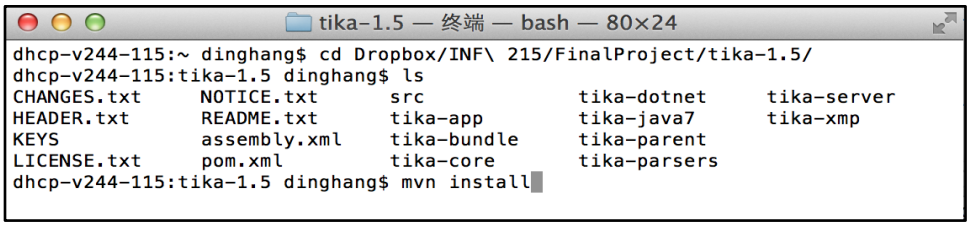


1.2.7 Type in “source ~/.bash_profile” in terminal to save the setting;

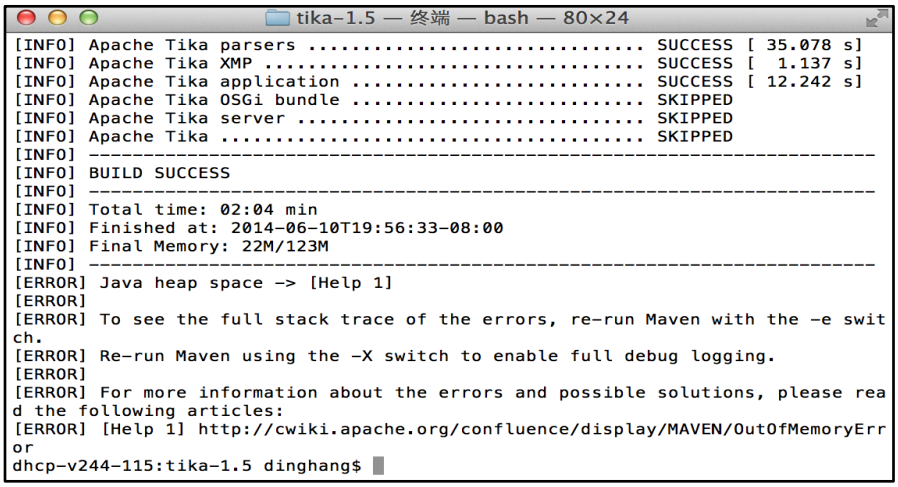
1.2.8 Use “mvn --version” in terminal to check that maven is working now;

1.3 Build Tika-1.5 with Maven;

1.3.1 Go to the folder where the uncompressed Tika-1.5 exists. In that folder, type in “mvn install” in terminal to build Tika-1.5;



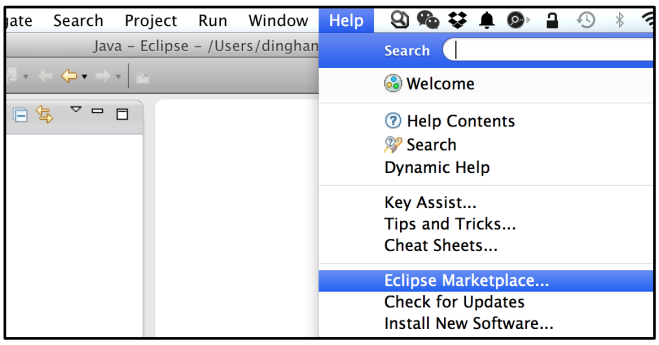
1.3.2 Press “enter”. Wait for a while. At last, you can see “BUILD SUCCESS”;



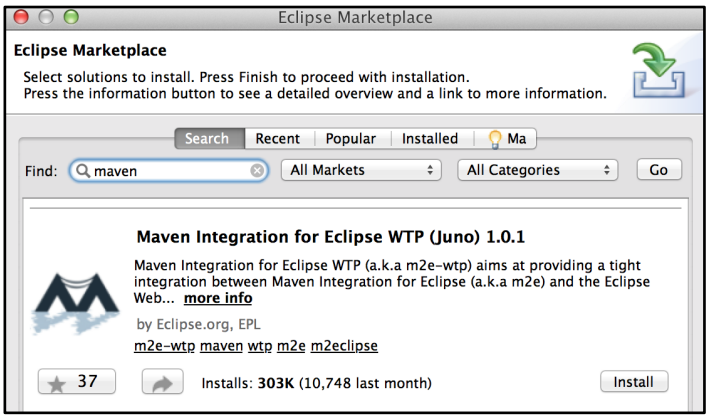
1.4 Import Tika-1.5 to Eclipse for testing;

1.4.1 Since tika-1.5 is a Maven project, so we need to install Maven plugin in eclipse;

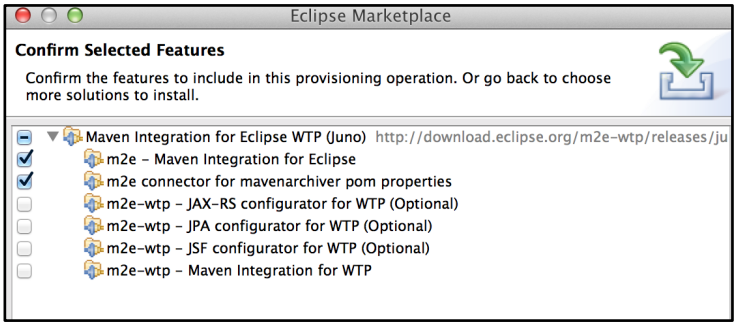
In Eclipse, click “help” icon on the menu bar, choose “Eclipse Marketplace”;



Search “Maven”, Choose “Maven Integration for Eclipse WTP(Juno) 1.0.1” to install;

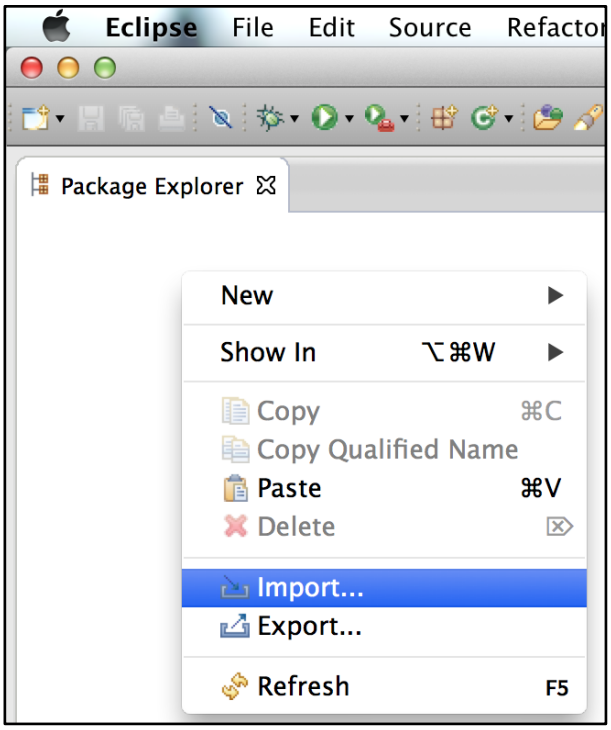


Choose install plugin as followed, then confirm this page, follow the process, choose “I accept the terms of the license agreements” before finish;

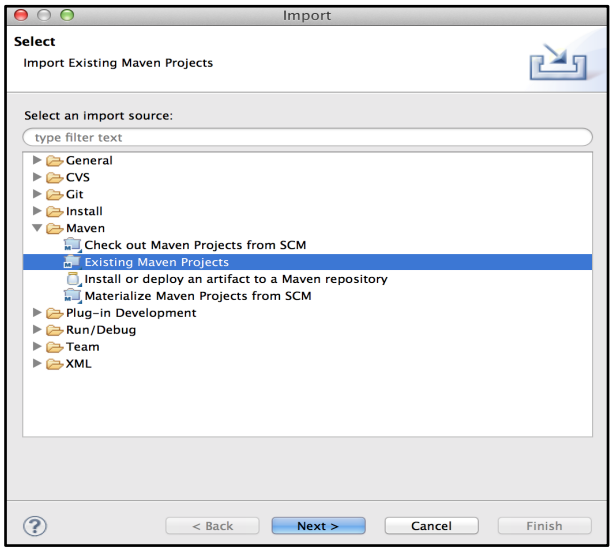


1.4.2. Import tika-1.5;

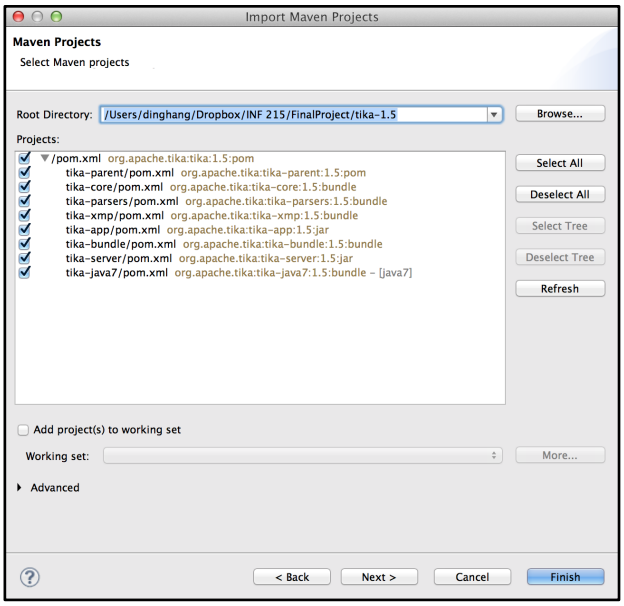
At the Package Explorer column, right click, choose “Import”;



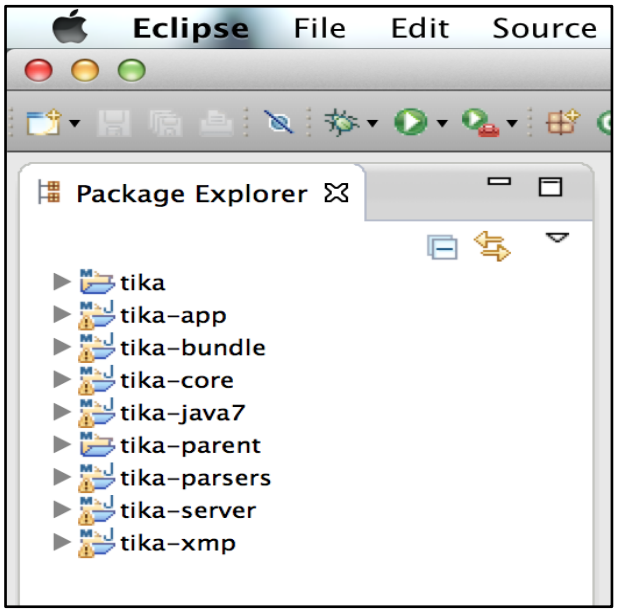
Choose “Existing Maven Project”;



Browse the root folder of the uncompressed tika-1.5, confirm, then click finish;



All set;

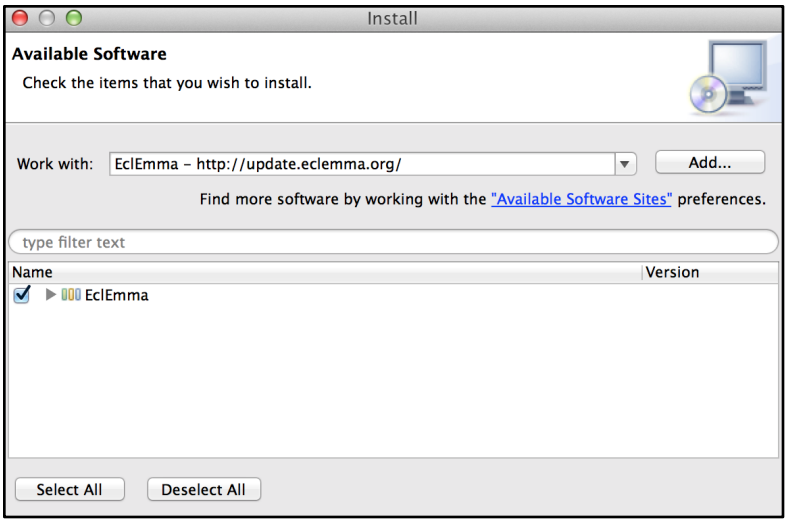


2. Preliminary Testing and Coverage

2.1 Install Eclemma

2.1.1 On the Eclipse menu bar, select Help -> Install New Software

2.1.2 In the Work With field, enter http://update.eclemma.org/



2.1.3 Check the box and press next

2.1.4 Follow the steps in the installation wizard

2.2. Run the given test cases and report the coverage

2.2.1 Results of test cases in tika-app

tika-app	48.4 %	2,068	2,204	4,272
src/main/java	41.0 %	1,524	2,196	3,720
org.apache.tika.gui	0.0 %	0	1,344	1,344
org.apache.tika.cli	64.1 %	1,524	852	2,376
TikaCLI.java	64.1 %	1,524	852	2,376

2.2.2 Results of test cases in tika-core

tika-core	63.9 %	25,123	14,213	39,336
src/main/java	48.7 %	12,757	13,464	26,221
org.apache.tika.io	36.5 %	1,180	2,051	3,231
org.apache.tika.metadata	51.2 %	1,732	1,651	3,383
org.apache.tika.sax	53.8 %	1,863	1,600	3,463
org.apache.tika.parser	17.5 %	263	1,240	1,503
org.apache.tika.fork	44.7 %	960	1,189	2,149
org.apache.tika.mime	69.8 %	2,611	1,132	3,743
org.apache.tika.parser.external	0.0 %	0	991	991
org.apache.tika.language	54.1 %	1,053	895	1,948
org.apache.tika.config	29.4 %	333	799	1,132
org.apache.tika.embedder	0.0 %	0	715	715
org.apache.tika.detect	81.0 %	1,515	355	1,870
org.apache.tika	16.6 %	64	321	385
org.apache.tika.extractor	0.0 %	0	317	317
org.apache.tika.sax.xpath	81.9 %	508	112	620
org.apache.tika.utils	89.5 %	666	78	744
org.apache.tika.exception	33.3 %	9	18	27

2.2.3 Results of test cases in tika-java7

tika-java7	98.6 %	137	2	139
src/main/java	94.1 %	32	2	34
org.apache.tika.filetypedetector	94.1 %	32	2	34
TikaFileTypeDetector.java	94.1 %	32	2	34

2.2.4 Results of test cases in tika-parser

tika-parsers	88.9 %	92,565	11,570	104,135
src/main/java	86.6 %	64,594	9,985	74,579
org.apache.tika.parser.iptc	2.8 %	55	1,934	1,989
org.apache.tika.parser.chm.lzx	64.4 %	2,395	1,324	3,719
org.apache.tika.parser.mp3	80.8 %	3,471	823	4,294
org.apache.tika.parser.microsoft	88.8 %	5,107	644	5,751
org.apache.tika.parser.pdf	64.2 %	1,118	623	1,741
org.apache.tika.parser.txt	98.1 %	27,619	528	28,147
org.apache.tika.parser.microsoft.ooxml	84.3 %	2,641	492	3,133
org.apache.tika.parser.chm.core	76.1 %	1,340	421	1,761
org.apache.tika.parser.rtf	89.0 %	2,954	365	3,319
org.apache.tika.parser.executable	58.9 %	514	358	872
org.apache.tika.parser.chm.accessor	90.7 %	3,165	324	3,489
org.apache.tika.parser.xml	76.1 %	905	285	1,190
org.apache.tika.parser.image	83.9 %	1,349	259	1,608
org.apache.tika.parser.pkg	84.9 %	904	161	1,065
org.apache.tika.parser.mail	70.1 %	371	158	529
org.apache.tika.parser.code	21.7 %	41	148	189
org.apache.tika.parser.odf	91.5 %	1,459	136	1,595
org.apache.tika.parser.iwork	95.4 %	2,778	135	2,913
org.apache.tika.parser.asm	79.4 %	477	124	601
org.apache.tika.parser.video	72.4 %	320	122	442
org.apache.tika.parser.chm.assertion	36.4 %	52	91	143
org.apache.tika.parser.html	96.1 %	2,184	89	2,273
org.apache.tika.parser.audio	78.7 %	295	80	375
org.apache.tika.parser.font	71.8 %	201	79	280
org.apache.tika.parser.crypto	27.5 %	22	58	80
org.apache.tika.parser.mbox	90.4 %	407	43	450
org.apache.tika.parser.internal	0.0 %	0	39	39
org.apache.tika.parser.epub	82.0 %	155	34	189
org.apache.tika.parser.image.xmp	90.2 %	221	24	245
org.apache.tika.parser.mp4	96.2 %	500	20	520
org.apache.tika.parser.dwg	97.2 %	584	17	601
org.apache.tika.parser.prt	97.4 %	451	12	463
org.apache.tika.parser.feed	94.8 %	164	9	173
org.apache.tika.parser.chm	94.9 %	130	7	137
org.apache.tika.parser.hdf	93.5 %	100	7	107
org.apache.tika.parser.netcdf	93.0 %	93	7	100
org.apache.tika.parser.jpeg	90.0 %	45	5	50
org.apache.tika.parser.chm.exception	100.0 %	4	0	4
org.apache.tika.parser.opendocument	100.0 %	3	0	3

2.2.5 Result of test cases in tika-server

tika-server	78.4 %	2,571	709	3,280
src/main/java	63.6 %	1,231	705	1,936
org.apache.tika.server	63.6 %	1,231	705	1,936
TikaServerCli.java	0.0 %	0	270	270
TikaResource.java	68.1 %	421	197	618
UnpackerResource.java	57.5 %	218	161	379
RichTextContentHandler.java	64.6 %	42	23	65
ZipWriter.java	81.2 %	78	18	96
MetadataEP.java	93.4 %	155	11	166
MetadataResource.java	91.7 %	111	10	121
JSONMessageBodyWriter.java	89.2 %	66	8	74
TikaExceptionMapper.java	83.3 %	15	3	18
CSVMessageBodyWriter.java	96.9 %	63	2	65
TarWriter.java	96.4 %	54	2	56
TikaVersion.java	100.0 %	8	0	8

2.2.6 Results of test cases in tika-xmp

tika-xmp	73.5 %	1,774	639	2,413
src/main/java	65.9 %	1,111	576	1,687
org.apache.tika.xmp.convert	58.6 %	625	441	1,066
MSOfficeBinaryConverter.java	0.0 %	0	133	133
OpenDocumentConverter.java	0.0 %	0	125	125
RTFConverter.java	0.0 %	0	80	80
MSOfficeXMLConverter.java	86.2 %	194	31	225
TikaToXMP.java	82.8 %	149	31	180
GenericConverter.java	80.7 %	109	26	135
AbstractConverter.java	91.6 %	164	15	179
Namespace.java	100.0 %	9	0	9
org.apache.tika.xmp	78.3 %	486	135	621
XMPMetadata.java	78.3 %	486	135	621

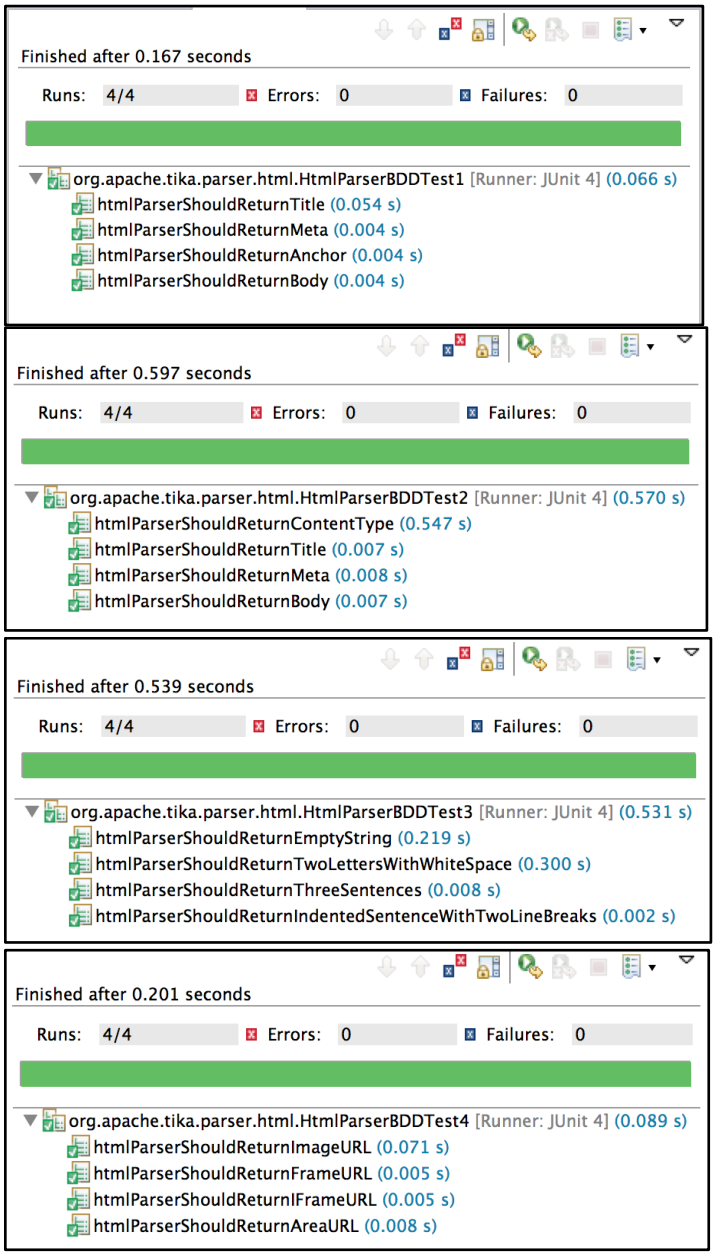
2. BDD (Functional Testing)

Test cases in BDD style

3.1 Test cases for HTMLParser class

3.1.1 To check source code, please check here: https://github.com/kanrourou/software-testing/tree/master/src/BDDTestcases/HtmlParserTestCases.

3.1.2 Running results



3.2 Test cases for ImageParser class

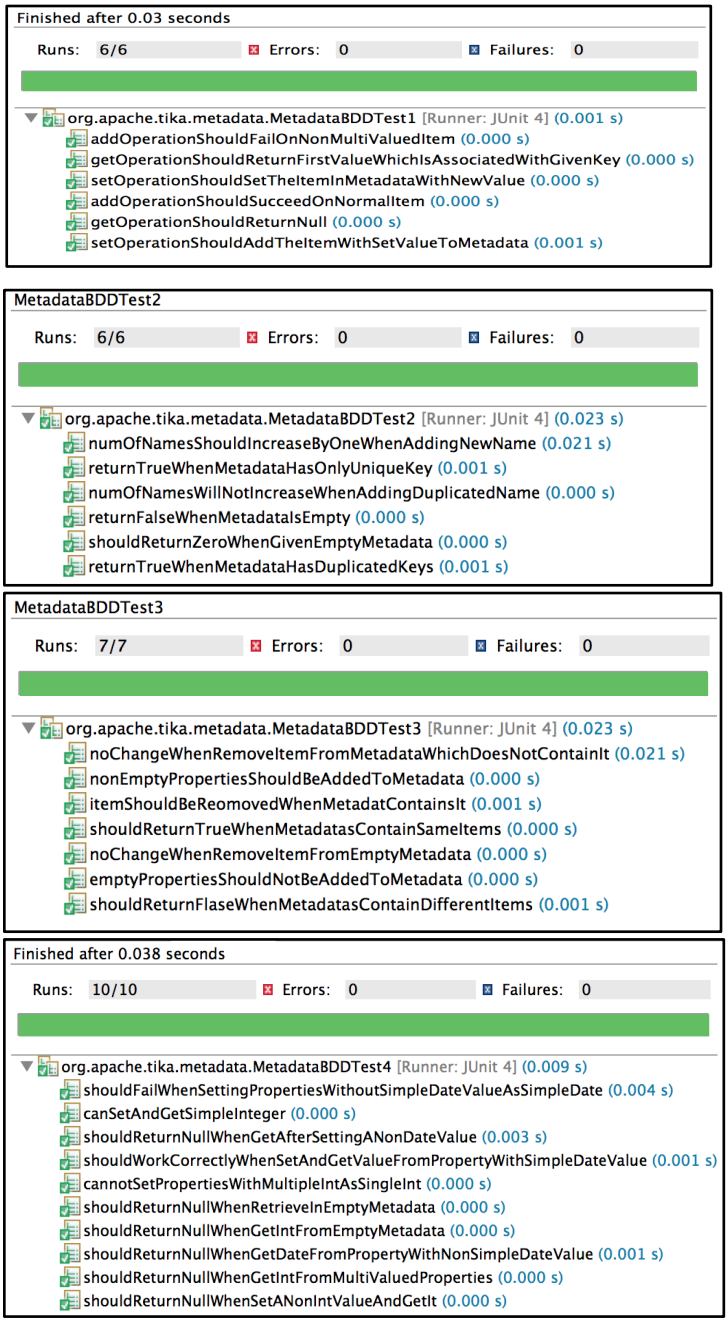
3.2.1 To check source code. please check here: https://github.com/kanrourou/software-testing/tree/master/src/BDDTestcases/ImageParserTestCases.

3.2.2 Running results

3.3 Test cases for Metadata class

3.3.1 To check source code, please check here: https://github.com/kanrourou/software-testing/tree/master/src/BDDTestcases/MetadataTestCases.

3.3.2 Running results



4. Static Code Analysis (Structural testing)

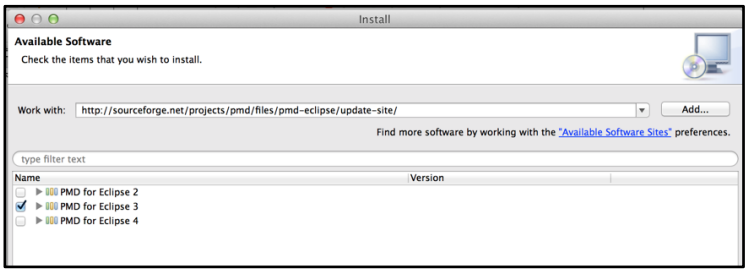
4.1 Install three different tools for static analysis

All of these tools could be plugged into the eclipse IDE, the way of installing them is similar. On the Eclipse menu bar, select Help -> Install New Software / Eclipse Marketplace

4.1.1 PMD

- Type into the Work With field in Install New Software tab with following stie

http://sourceforge.net/projects/pmd/files/pmd-eclipse/update-site/

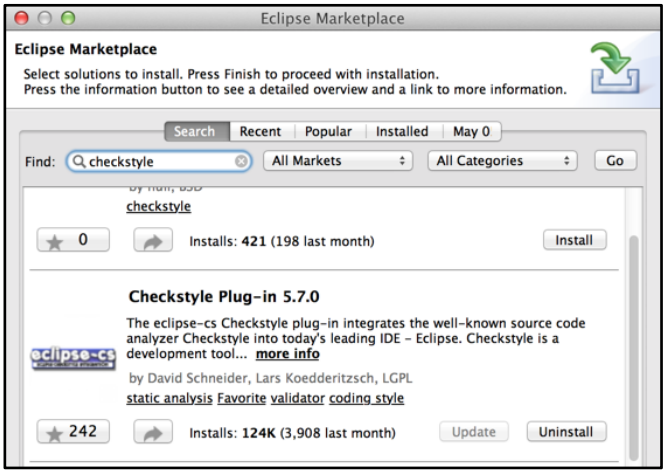


- Check the box and press next

- Follow the steps in the installation wizard

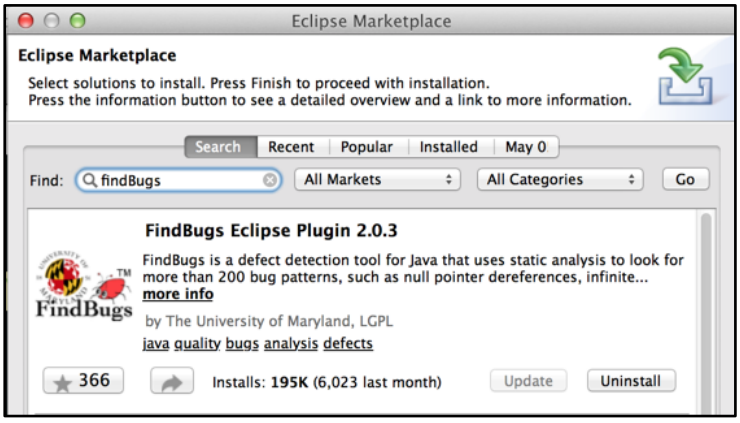
4.1.2 Checkstyle

In the Eclipse Marketplace tab search for Checkstyle, choose Checkstyle Plugin 5.7.0



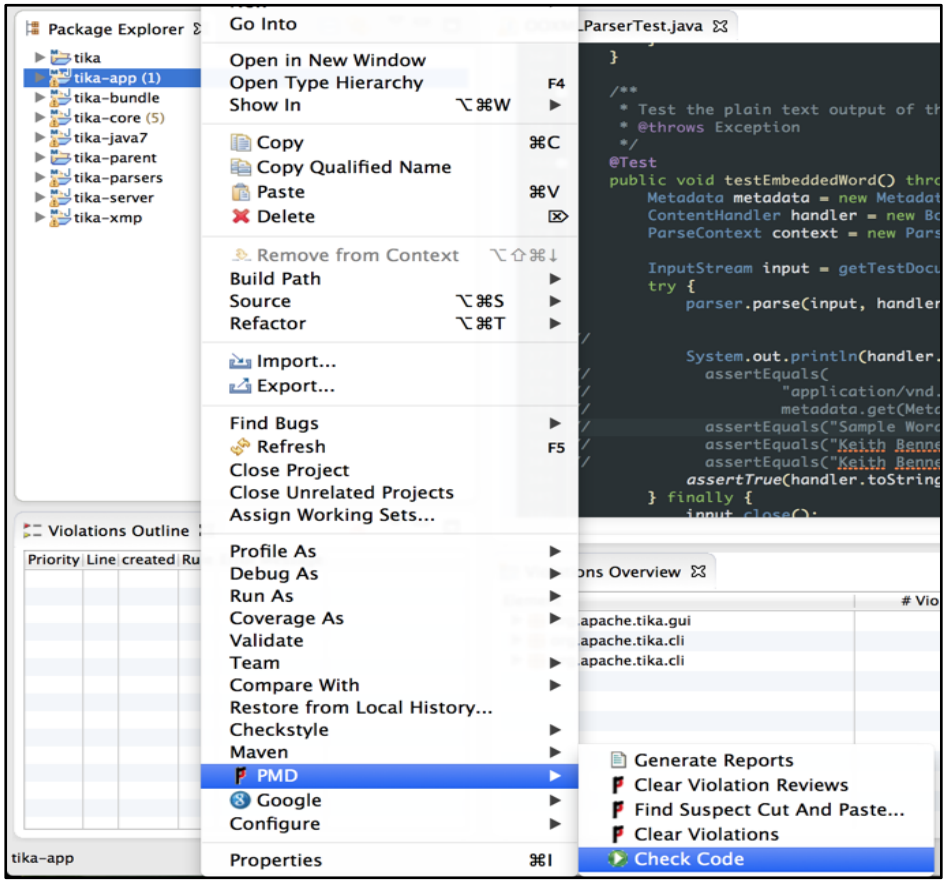
4.1.3 FindBugs

In the Eclipse Marketplace tab search for FindBugs, choose FindBugs Eclipse Plugin 2.0.3

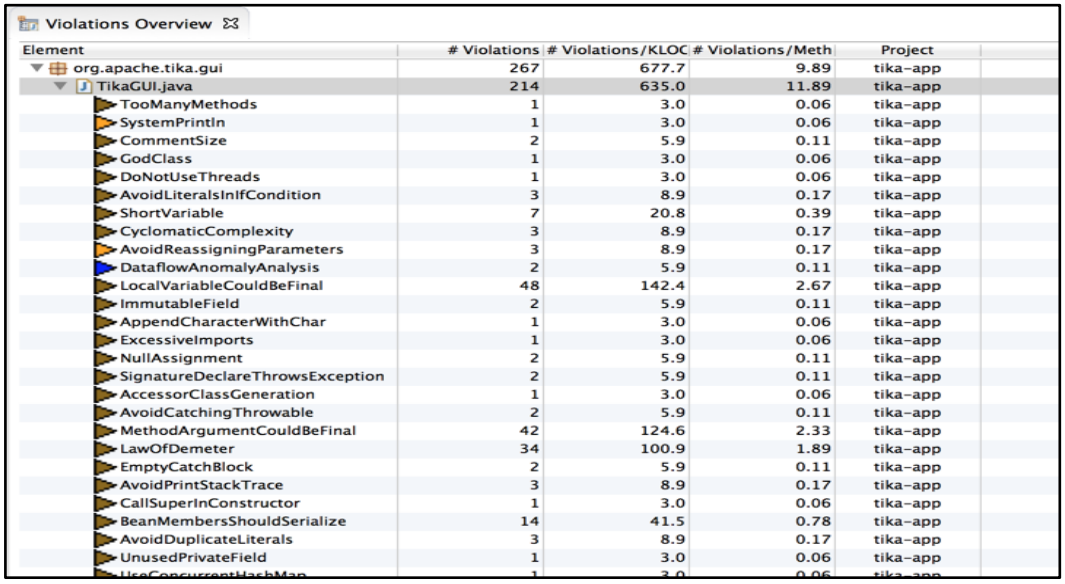


4.2 PMD Result (part of the check result)

To analyze the program by PMD, right click the project, and choose PMD -> check code, and the PMD analyzer will start to analyze the code.

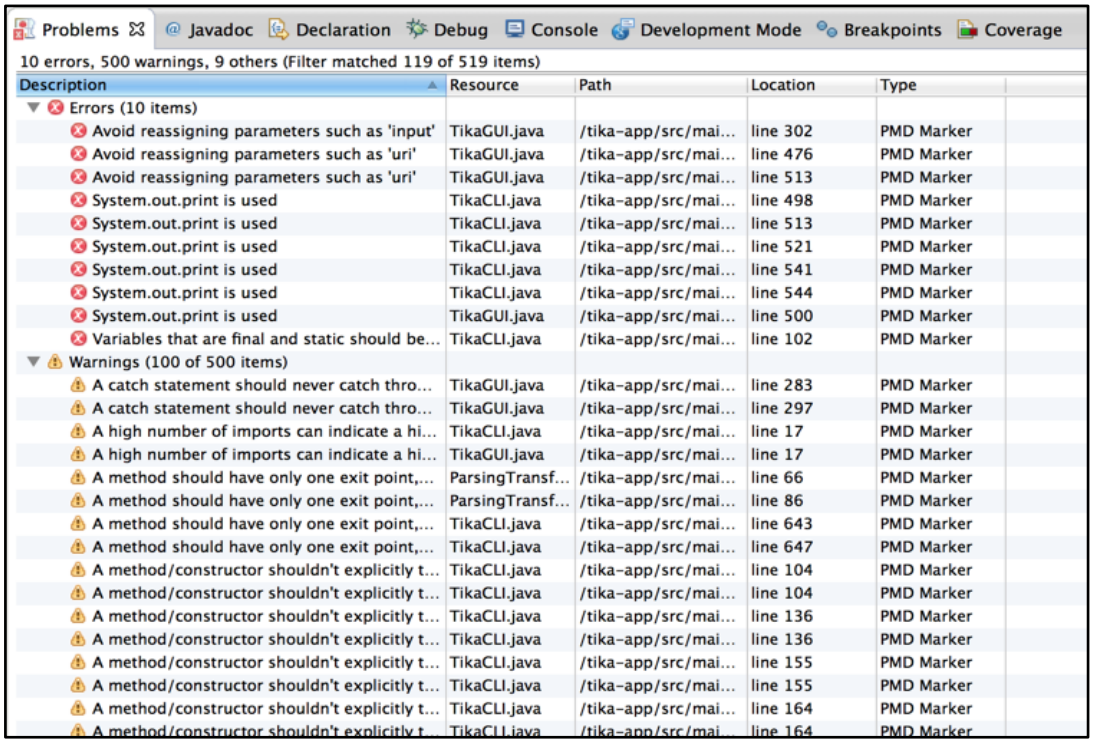


The result could be viewed in the Violations Overview window.

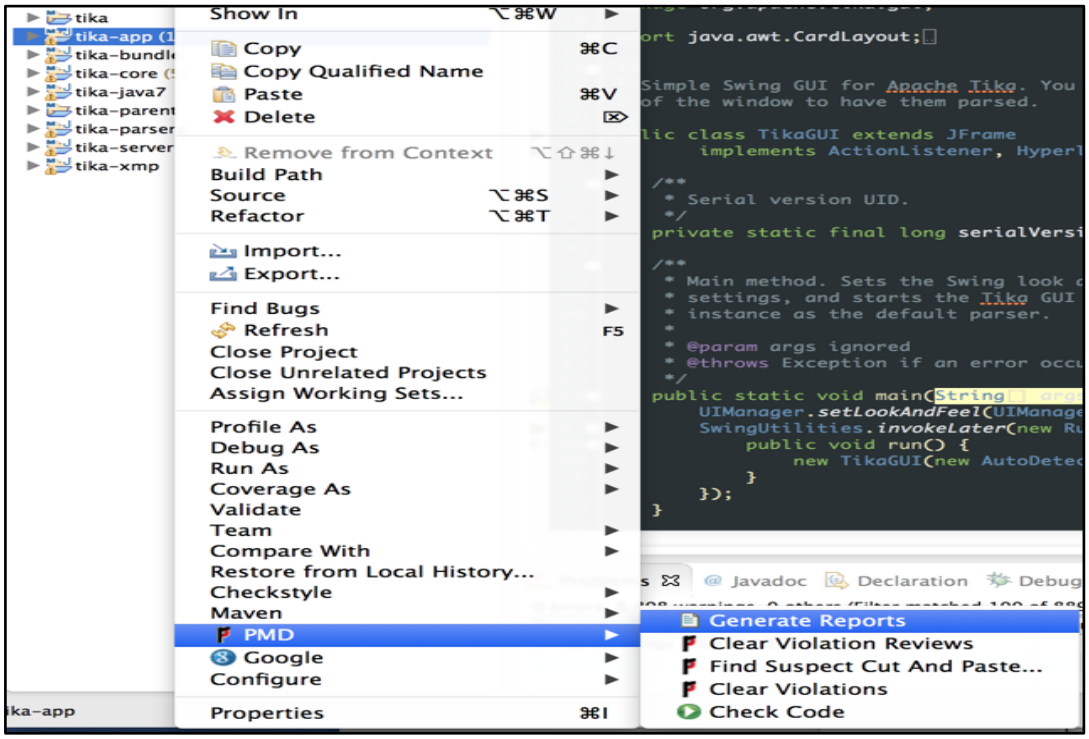


Element	# Violations	# Violations/KLOC	# Violations/Meth	Project
org.apache.tika.gui	267	677.7	9.89	tika-app
TikaGUI.java	214	635.0	11.89	tika-app
TooManyMethods	1	3.0	0.06	tika-app
SystemPrintln	1	3.0	0.06	tika-app
CommentSize	2	5.9	0.11	tika-app
GodClass	1	3.0	0.06	tika-app
DoNotUseThreads	1	3.0	0.06	tika-app
AvoidLiteralsInIfCondition	3	8.9	0.17	tika-app
ShortVariable	7	20.8	0.39	tika-app
CyclomaticComplexity	3	8.9	0.17	tika-app
AvoidReassigningParameters	3	8.9	0.17	tika-app
DataflowAnomalyAnalysis	2	5.9	0.11	tika-app
LocalVariableCouldBeFinal	48	142.4	2.67	tika-app
ImmutableField	2	5.9	0.11	tika-app
AppendCharacterWithChar	1	3.0	0.06	tika-app
ExcessiveImports	1	3.0	0.06	tika-app
NullAssignment	2	5.9	0.11	tika-app
SignatureDeclareThrowsException	2	5.9	0.11	tika-app
AccessorClassGeneration	1	3.0	0.06	tika-app
AvoidCatchingThrowable	2	5.9	0.11	tika-app
MethodArgumentCouldBeFinal	42	124.6	2.33	tika-app
LawOfDemeter	34	100.9	1.89	tika-app
EmptyCatchBlock	2	5.9	0.11	tika-app
AvoidPrintStackTrace	3	8.9	0.17	tika-app
CallSuperInConstructor	1	3.0	0.06	tika-app
BeanMembersShouldSerialize	14	41.5	0.78	tika-app
AvoidDuplicateLiterals	3	8.9	0.17	tika-app
UnusedPrivateField	1	3.0	0.06	tika-app

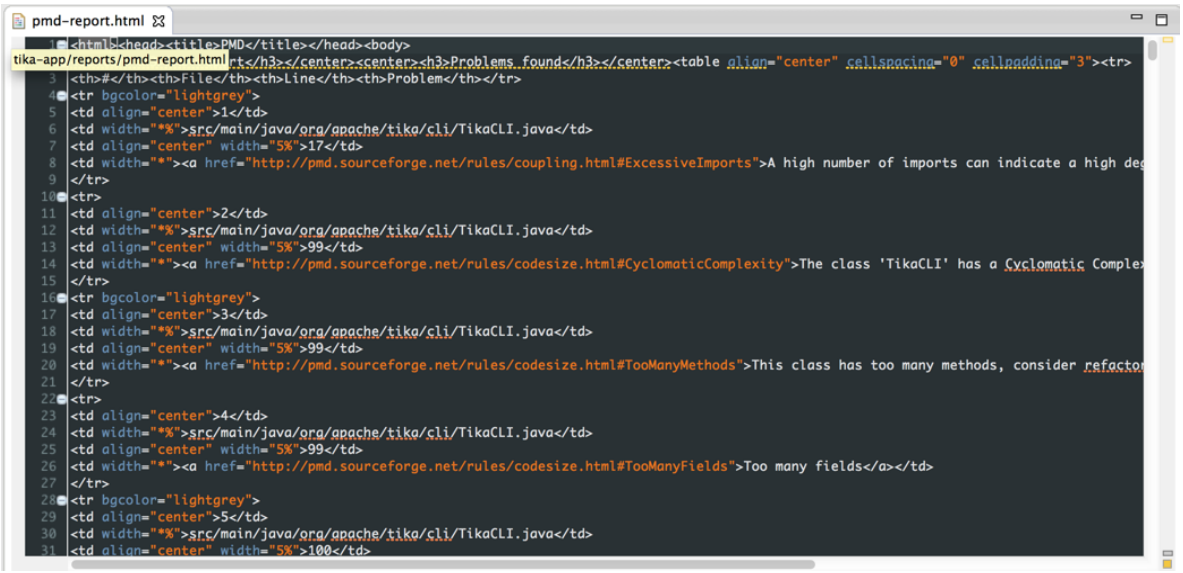
Also we could view the errors and warnings in the general Problem window.



To get the documented report for the checking, right click the project you want to report and choose PMD -> Generate Reports



There will be reports of the same content in different format, the screenshot of the html version of result is as follows



4.2.1 To check results of tika-app, please check here:

https://github.com/kanrourou/software-testing/blob/master/code static analyzer/report/PMD/tika_app_pmd.xml

4.2.2 To check results of tika-core, please check here:

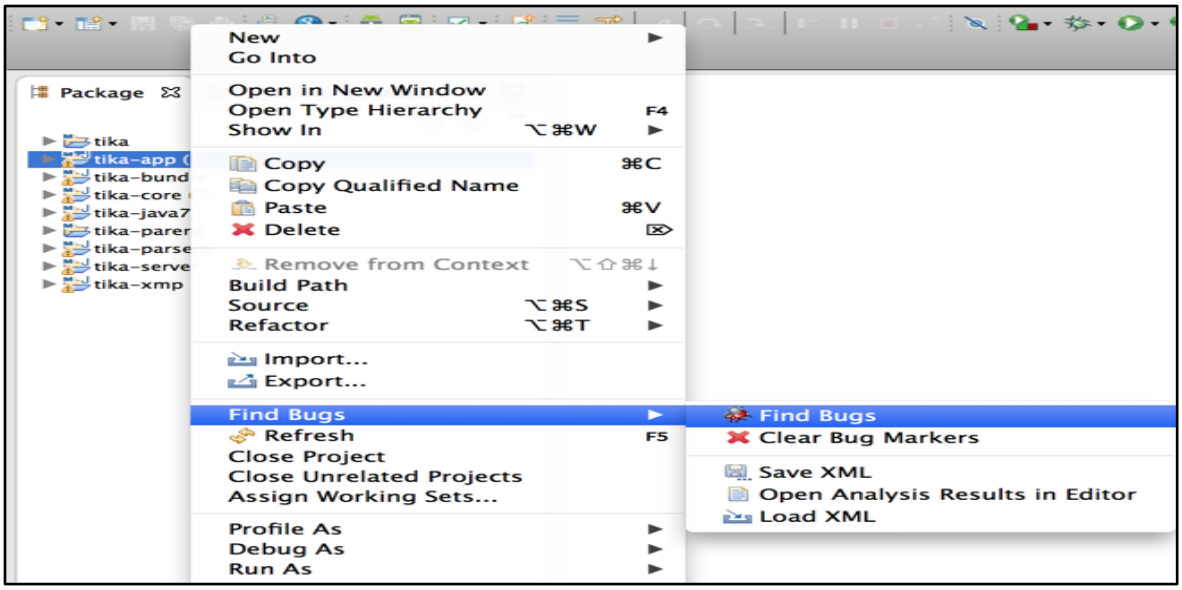
https://github.com/kanrourou/software-testing/blob/master/code static analyzer/report/PMD/tika_core_pmd.xml

4.2.3 To check results of tika-server, please check here:

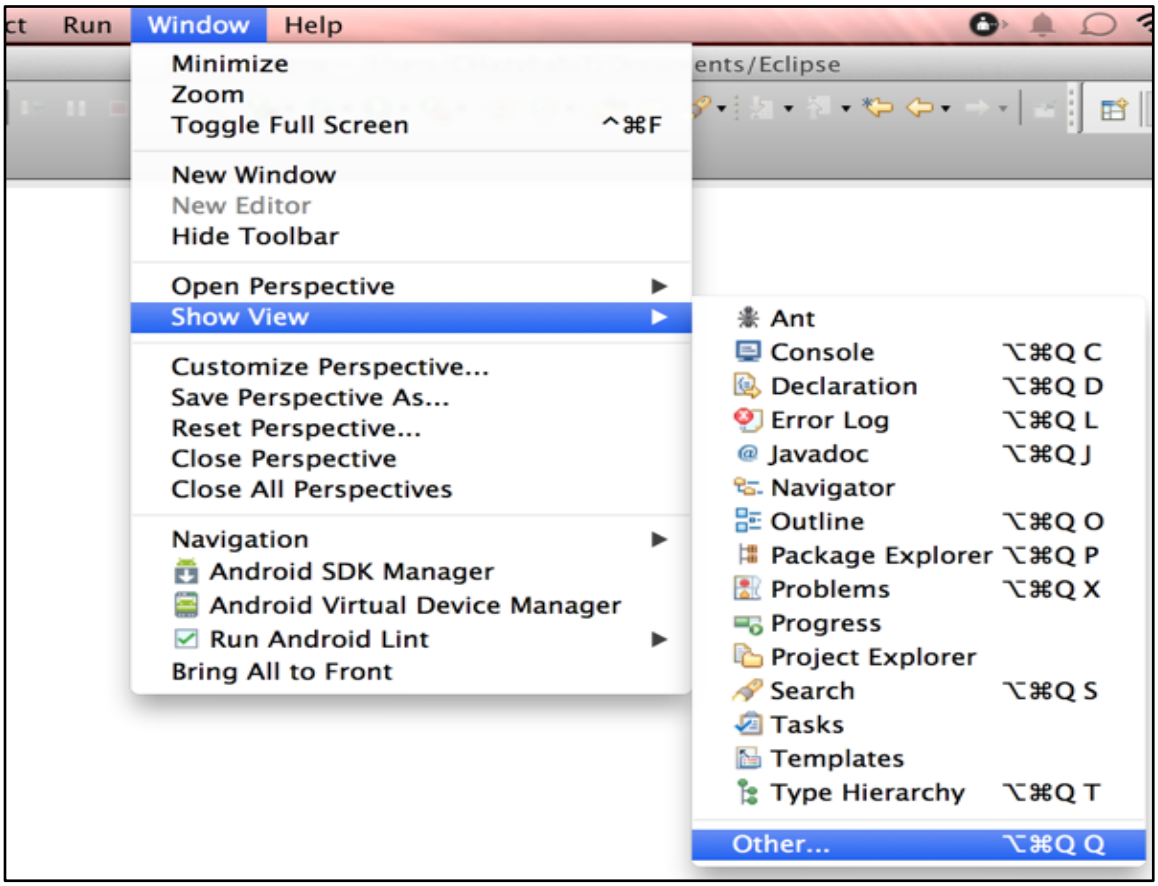
https://github.com/kanrourou/software-testing/blob/master/code static analyzer/report/PMD/tika_server_pmd.xml

4.3 FindBugs result (part of the check result)

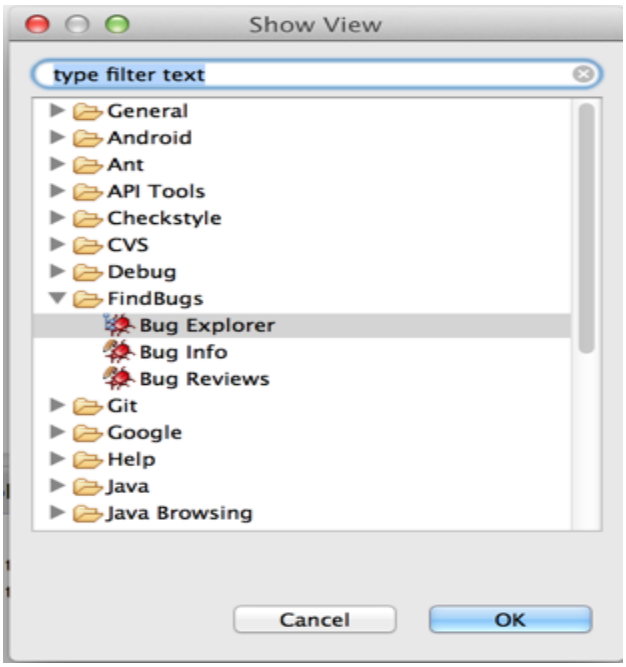
To get the checking results generated by FindBugs, right click the project, choose Find Bugs -> Find Bugs



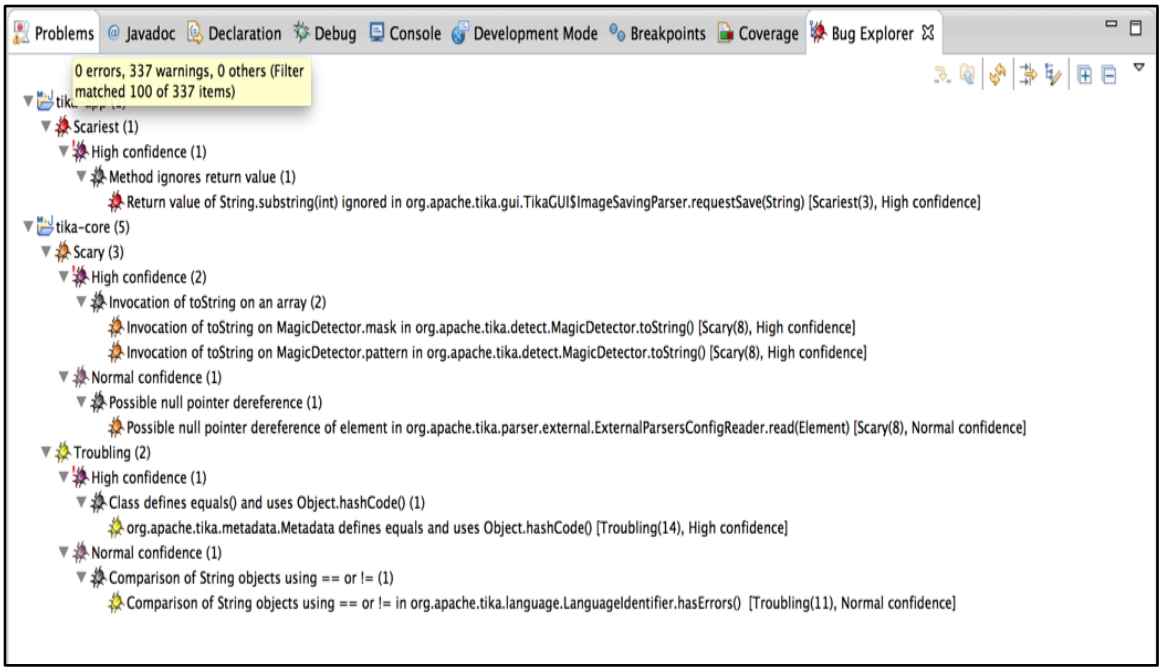
To view the result, we need to use the Bug Explore window. In the state of eclipse being focused, choose Window -> Show View -> Others...



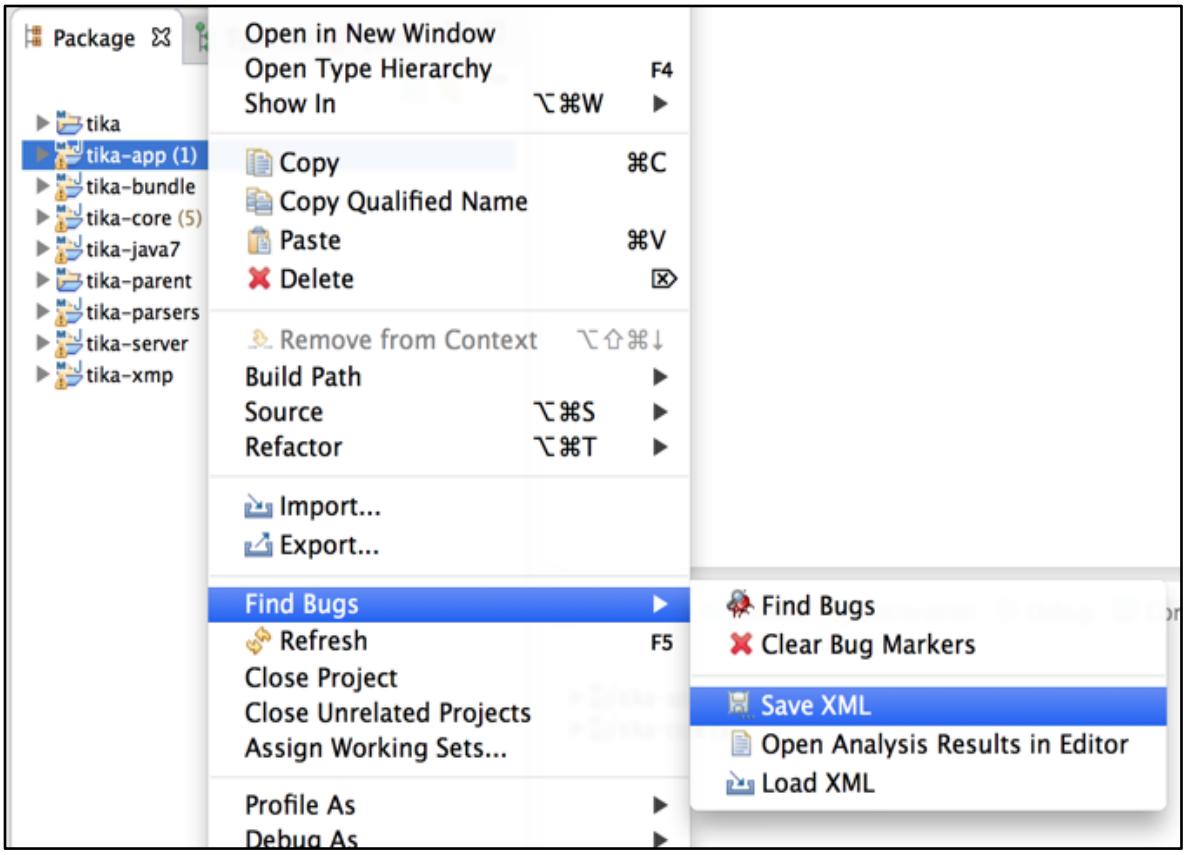
In the pop-up dialog, choose FindBugs -> Bug Explorer



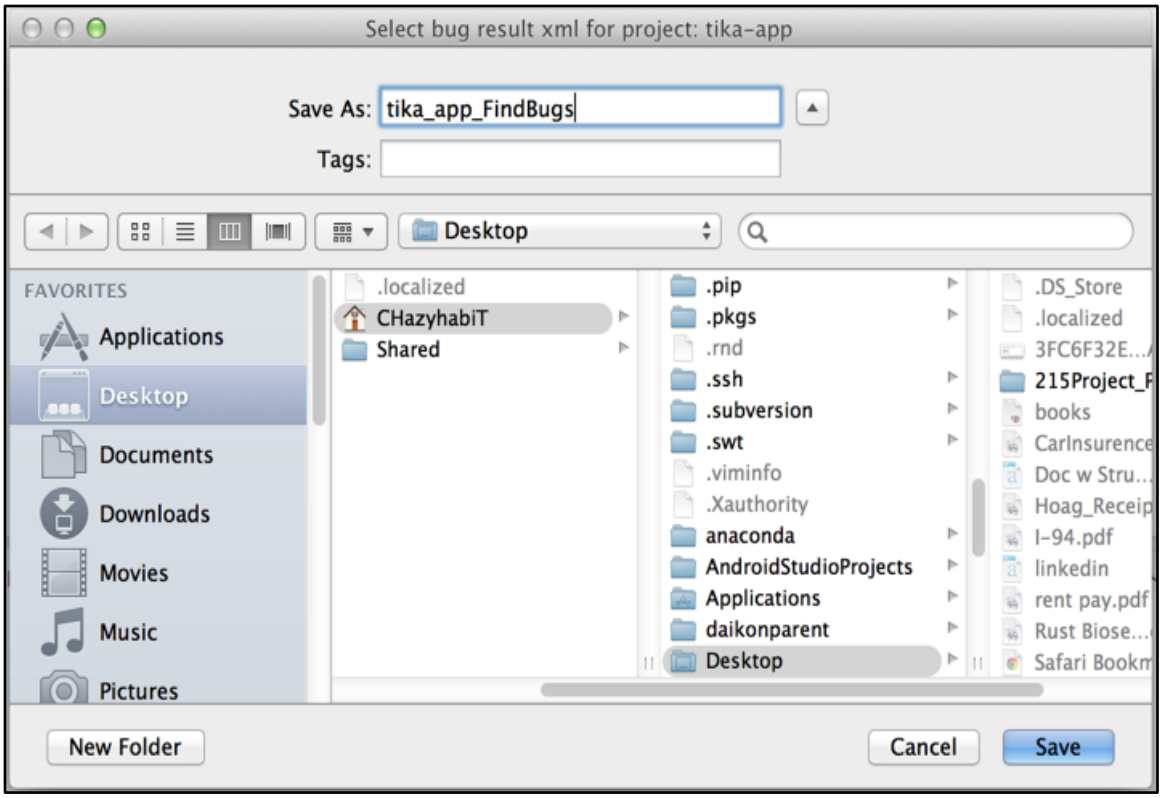
And we can see the result in the Bug Explorer window as follows



To save the result as a document, right click the project, choose Find Bugs -> Save XML



In the pop-up dialog, take a name and save it. Then we get the result report in the XML version.



4.3.1 To check the results of tika-app, please check here:

https://github.com/kanrourou/software-testing/blob/master/code static analyzer/report/FindBugs/tika_app_FindBugs

4.3.2 To check the results of tika-core, please check here:

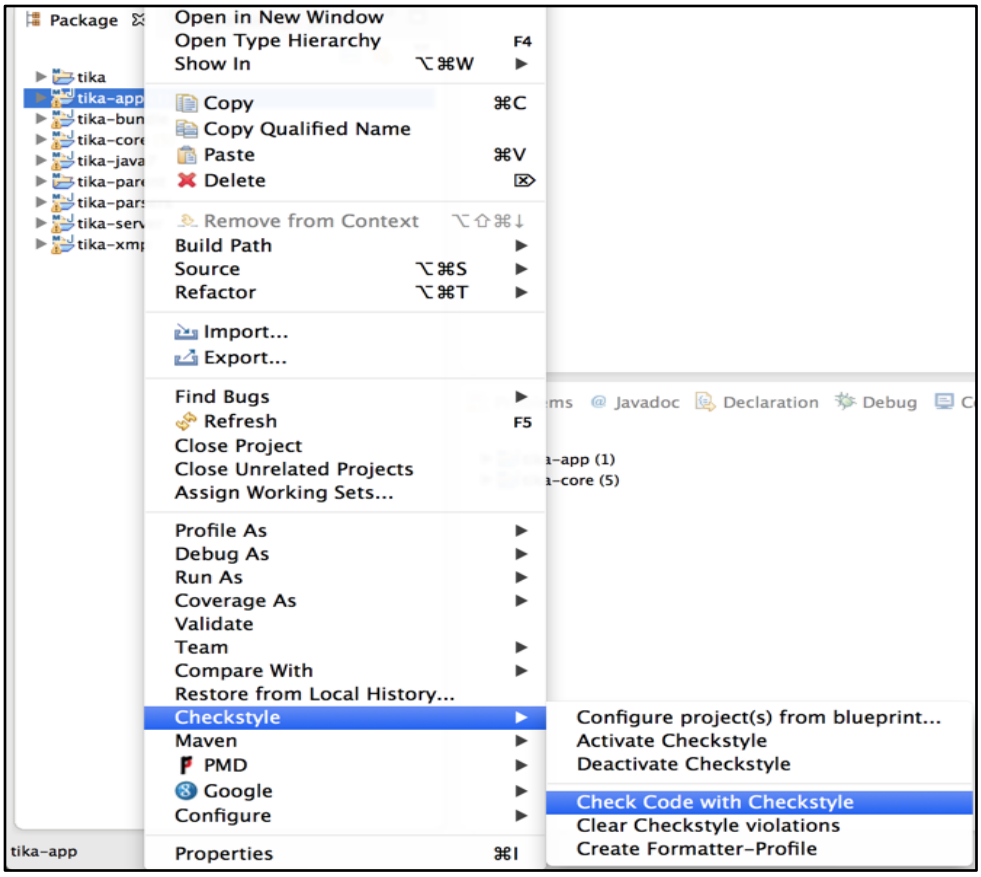
https://github.com/kanrourou/software-testing/blob/master/code static analyzer/report/FindBugs/tika_app_FindBugs

4.3.3 To check the results of tika-server, please check here:

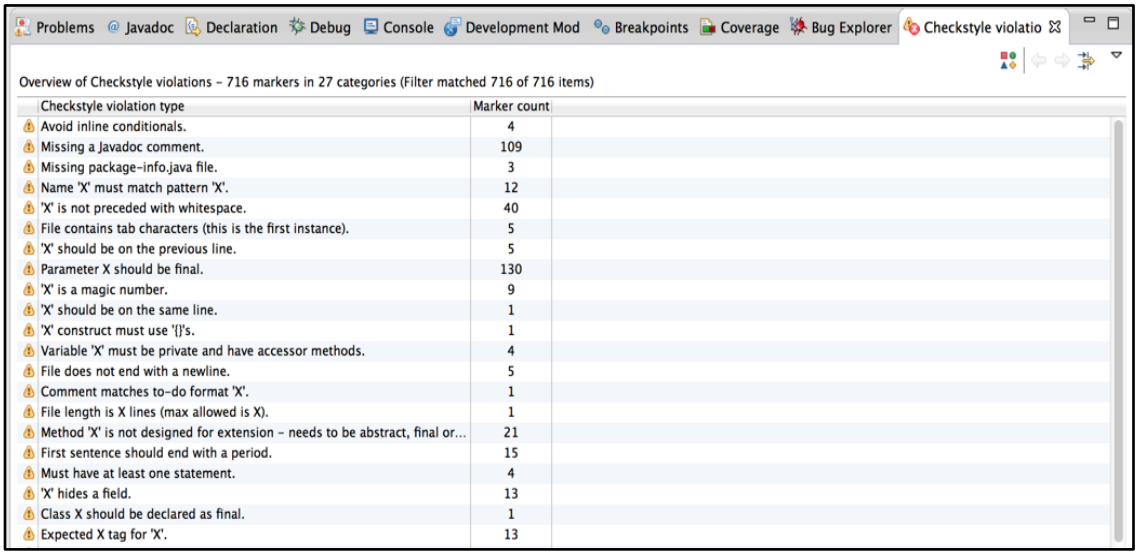
https://github.com/kanrourou/software-testing/blob/master/code static analyzer/report/FindBugs/tika_server_FindBugs

4.4 CheckStyle result (part of the check result)

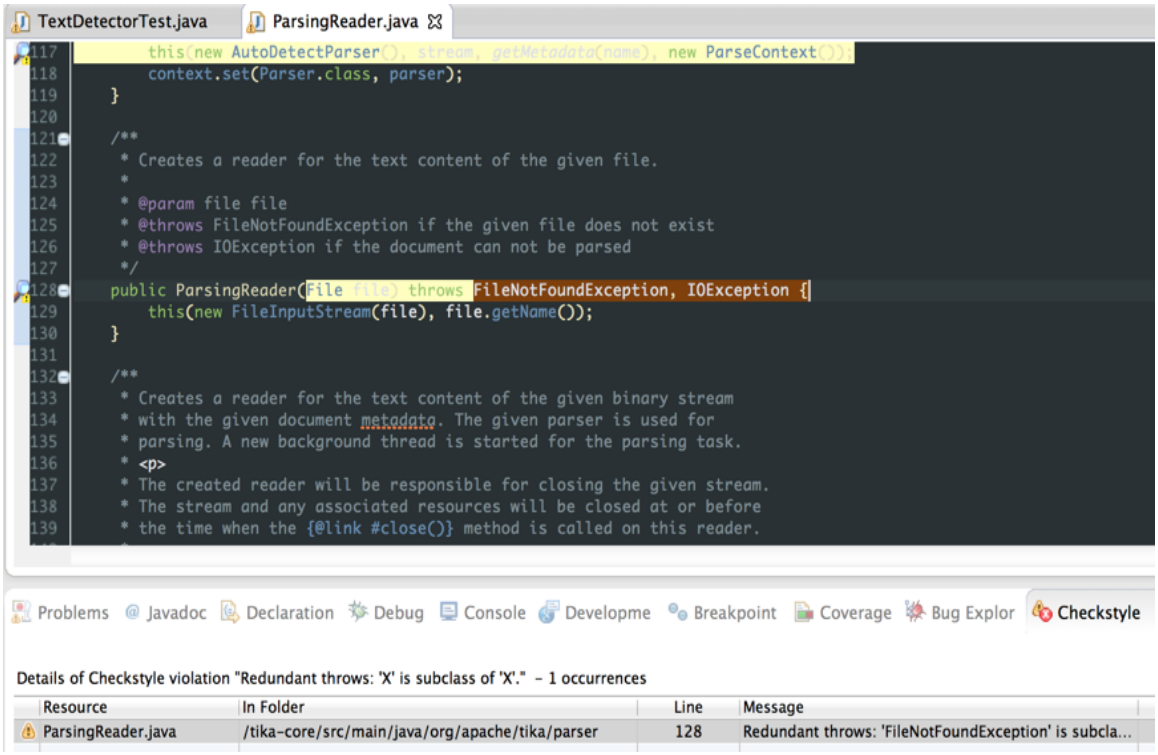
To get the result generated by the CheckStyle, right click the project, choose Checkstyle -> Check Code with Checkstyle.



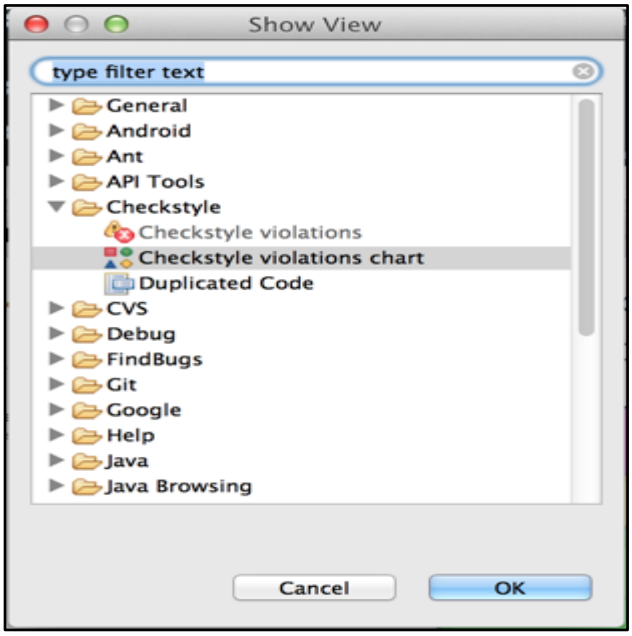
We can view the checking result in different way, the simplest way is to use the general Problem window, or use the Checkstyle violation window provided by the Checkstyle tool. The result is as follows:



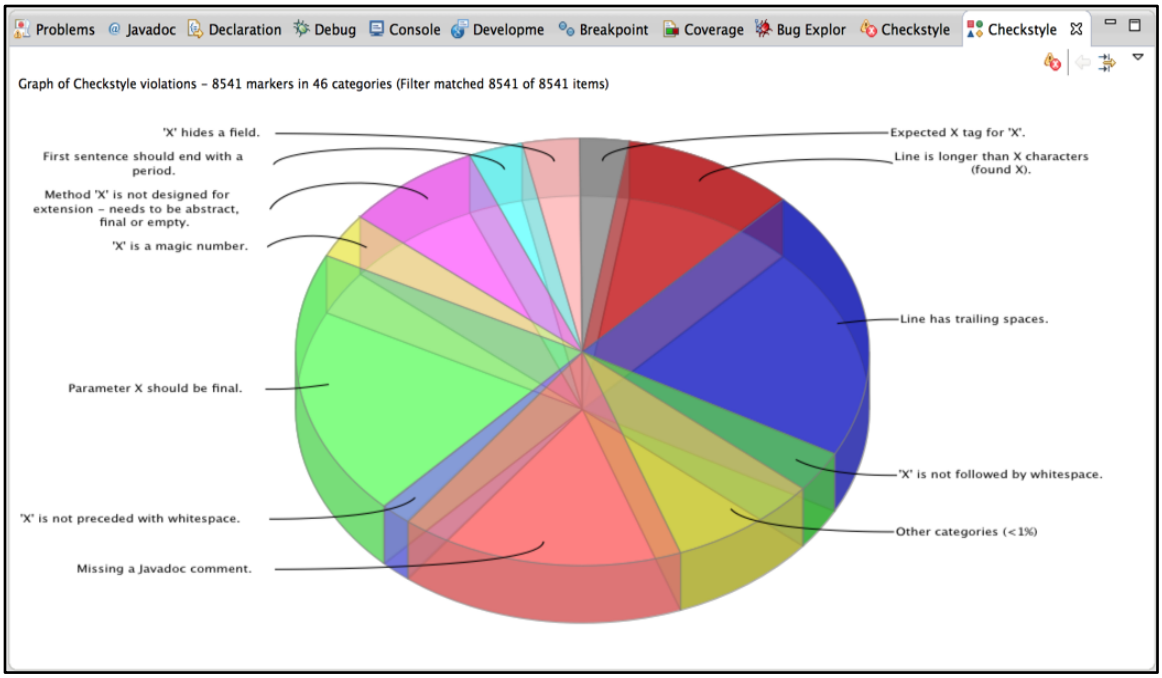
Click on the specific violation item, it will jump to the correspondence line of the program



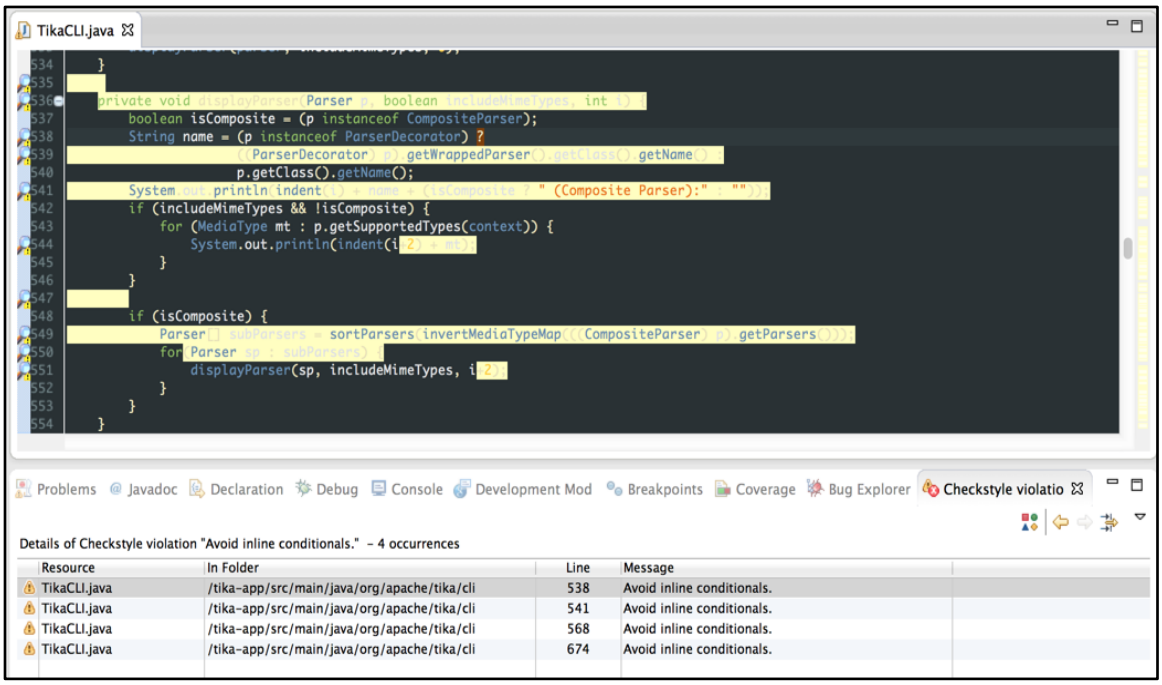
Moreover, if we want to view the result in statistic way, in the status of eclipse being focused, we can choose Window -> Show Views -> Others (as before), and in the pop-up dialog, choose Checkstyle -> Checkstyle Violation Chart



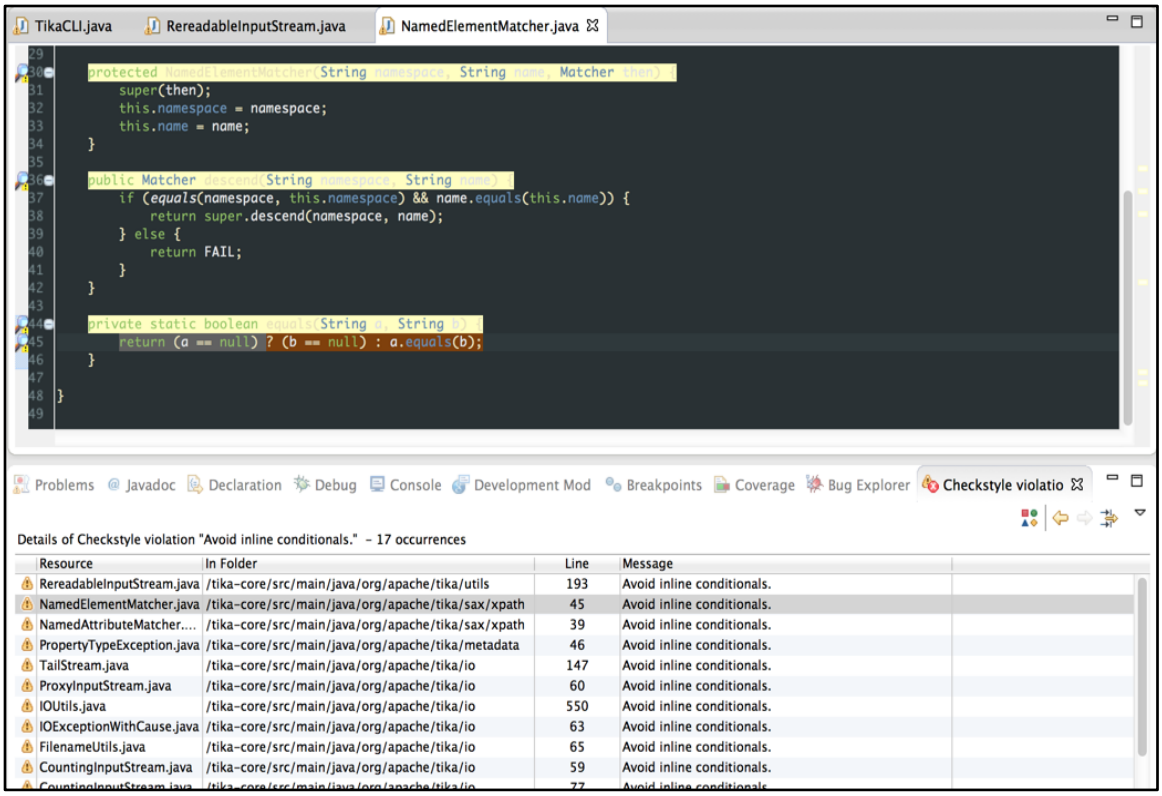
Then we can view the statistical results as follows:



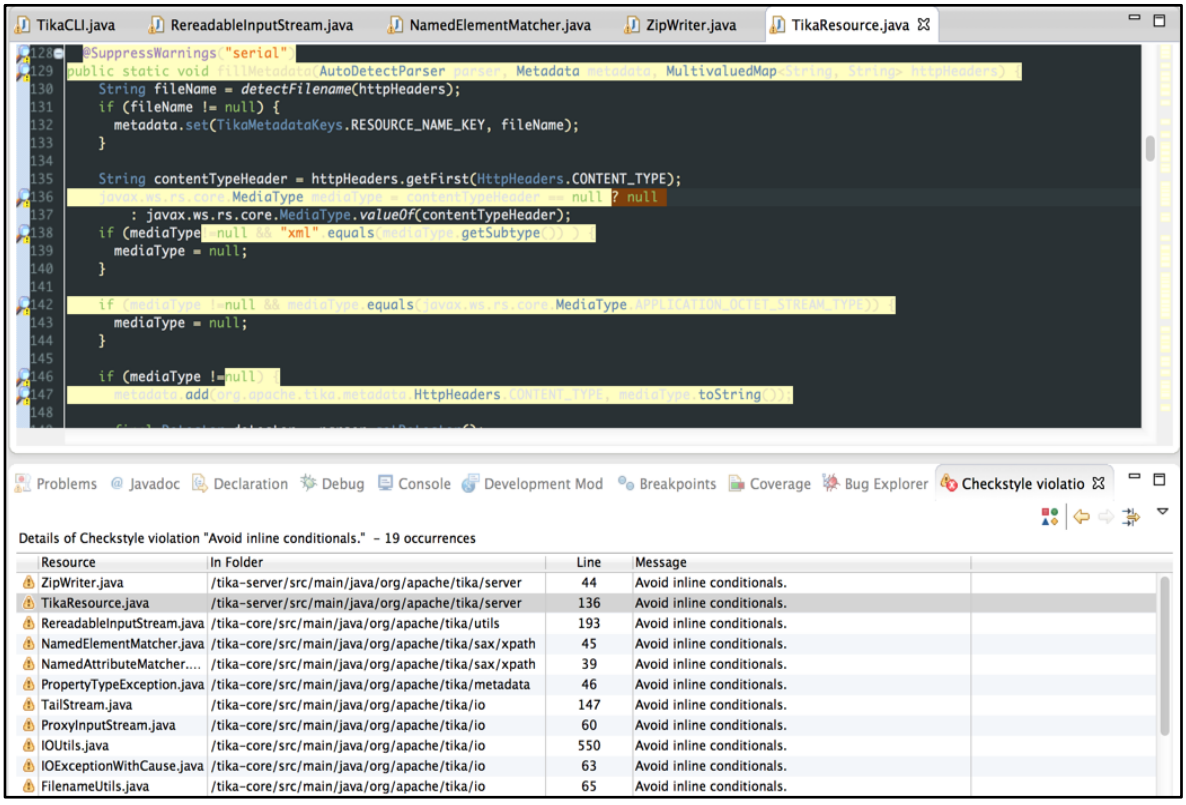
4.4.1 tika-app



4.4.2 tika-core



4.4.3 tika-server



4.5 Comparison among PMD, FindBugs, CheckStyle

PMD, FindBugs and CheckStyle, all of them could be used as the static analyzer for java program. They performs syntactic checks on program source code. The results of these analyzer could be exploited by many ways, such as the XML format, HTML format. In addition, they provide a much more convenient way for developer/tester to look up results in the development environment (IDE Plugins). However, each of them focuses on a specific area and each of them has its strength as well as its weaknesses. It is useful to have a comprehensive knowledge about their difference and focus in functionalities.

4.5.1 PMD

PMD is a static rule set based analyzer that identify potential risks. It looks for potential problems, possible bugs, unused and sub-optimal code and over complicated expressions in the Java source code. In our project, we run PMD with the ruleset defaultly recommended by the documentation (unusedcode.xml, basic.xml, import.xml, and favorites.xml). And the number of the warnings can increase or decrease depending on which ruleset we use. In some cases like the detection of clearly erroneous code, many of bus PMD regards as stylistic conventions whose violation might be suspicious under some circumstances.

Rule Categories:

- Possible bugs - Empty try/catch/finally/switch block.

- Dead code - Unused local variables, parameters and private methods

- Empty if/while statements

- Over complicated expressions - Unnecessary if statements, for loops, while loops.

- Suboptimal code - Wasteful String/ StringBuffer usage

Strength:

- finds occasional real defects

- find bad practices

Weakness:

- slow duplicate code detector

4.5.2 FindBugs

FindBugs looks for bugs in Java code. As the name itself suggests, it find bugs in Java bytecode. It uses static analysis to identify hundreds of different potential types of errors in Java

programs. It uses a series of ad-hoc techniques designed to balance precision, efficiency, and usability. One of the main techniques FindBugs uses is to syntactically match source code to known suspicious programming practice. In some cases, (Unlike to PMD) FindBugs also uses dataflow analysis to check for bugs. FindBugs can also be expanded by writing custom bug detectors in Java.

Rule Categories:

- Correctness
- Bad Practice
- Dodgy code
- Multithreaded Correctness
- Performance Malicious
- Code Vulnerability
- Security Experimental
- Internationalization

Strength:

- finds often real defects
- low false detection rates
- fast because byte code

Weakness:

- is not aware of the source
- needs compiled code

4.5.3 CheckStyle

CheckStyle is a development tool to help programmers write Java code that adheres to coding standard. CheckStyle defines a set of available modules, each of which provides rules checking with a configurable level of strictness (mandatory, optional), Each rule can raise notifications, warning, and errors. It scans source code and looks for coding standards like the Sun Code Conventions, JavaDoc.

Rule Categories:

- Annotations

- Block Checks

- Class Design

- Duplicate Code

- Headers

- Imports

- Modifier

- White space

Strength:

- finds violations of coding conventions

Weakness:

- cannot find real bugs

5. Reverse Engineering (Structural Testing)

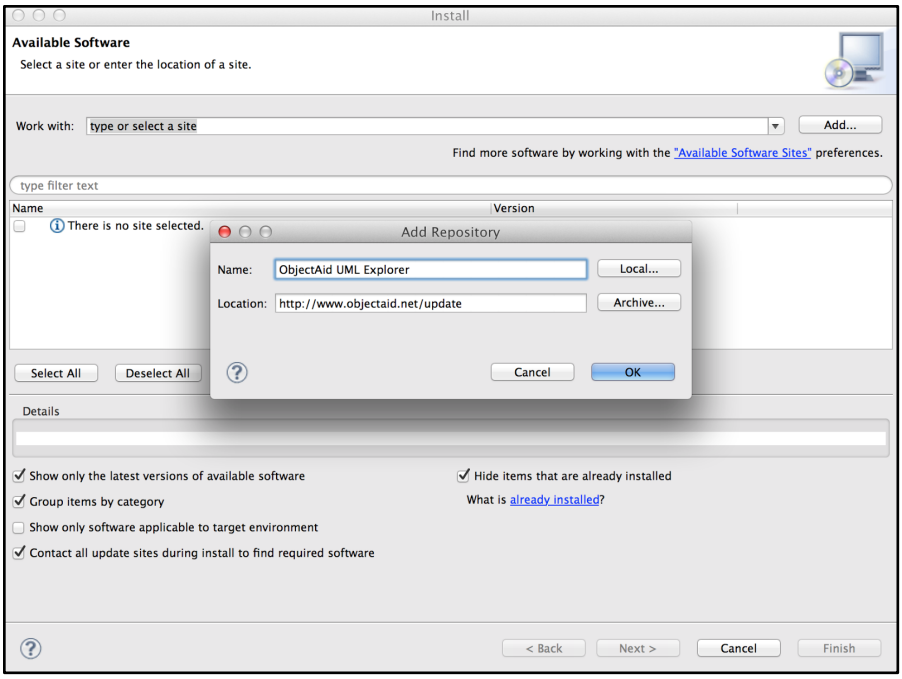
We have used three reverse-engineering tools to reveal the code structure respectively. ObjectAid for UML Class Diagram. Call Graph Viewer for Call Graph. Control Flow Graph Factory for Control Flow Graph.

5.1. UML Class Diagram

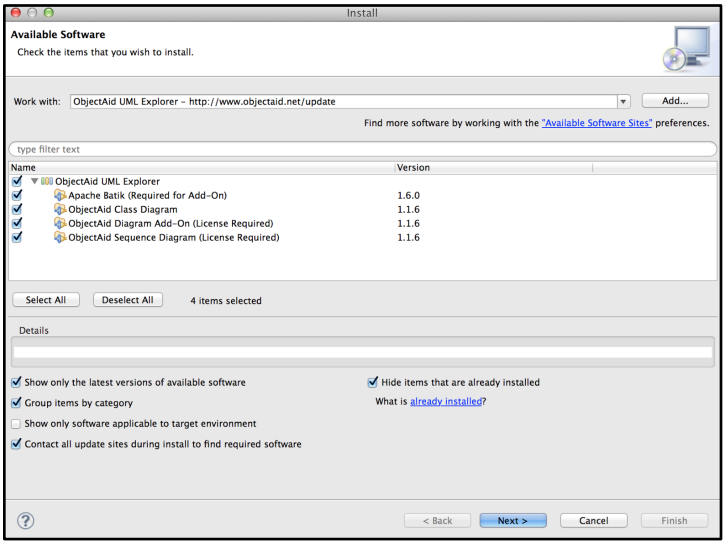
5.1.1 Installation;

- On the Eclipse menu bar, select Help -> Install New Software;

- Press "Add". Name: "ObjectAid UML Explorer". URL: "http://www.objectaid.net/update". Press OK;



- Check the box and press next;

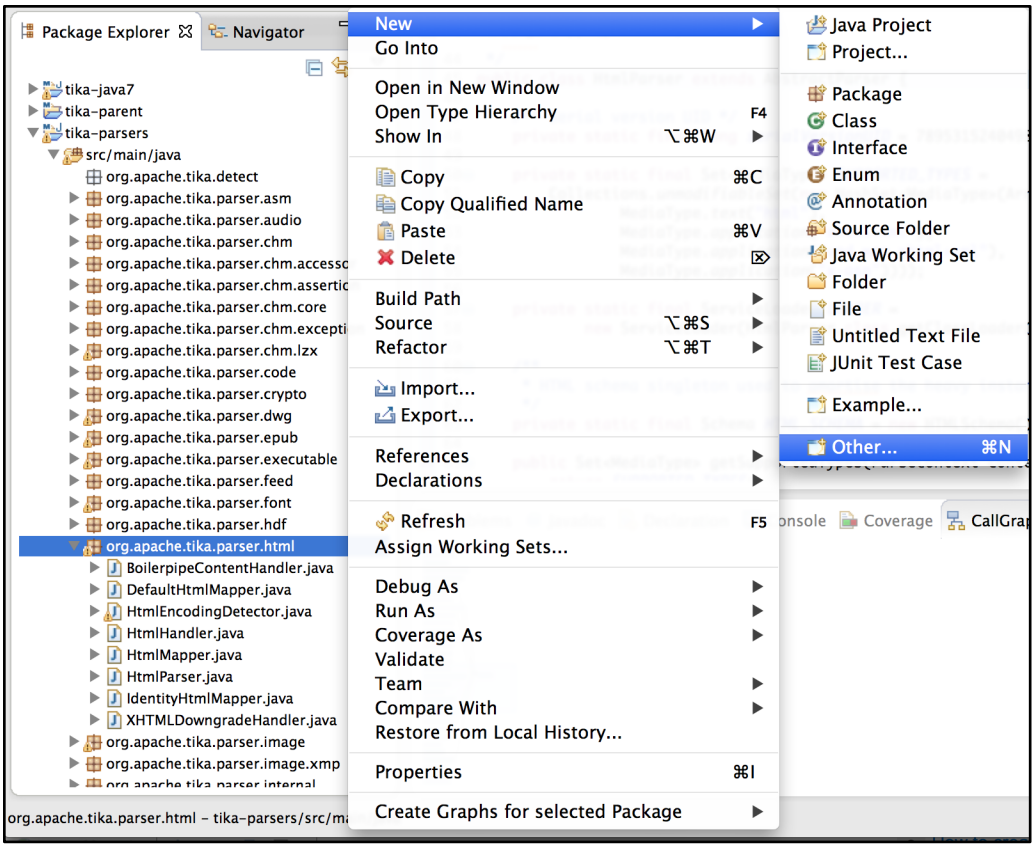


- Follow the steps in the wizard to finish installation;

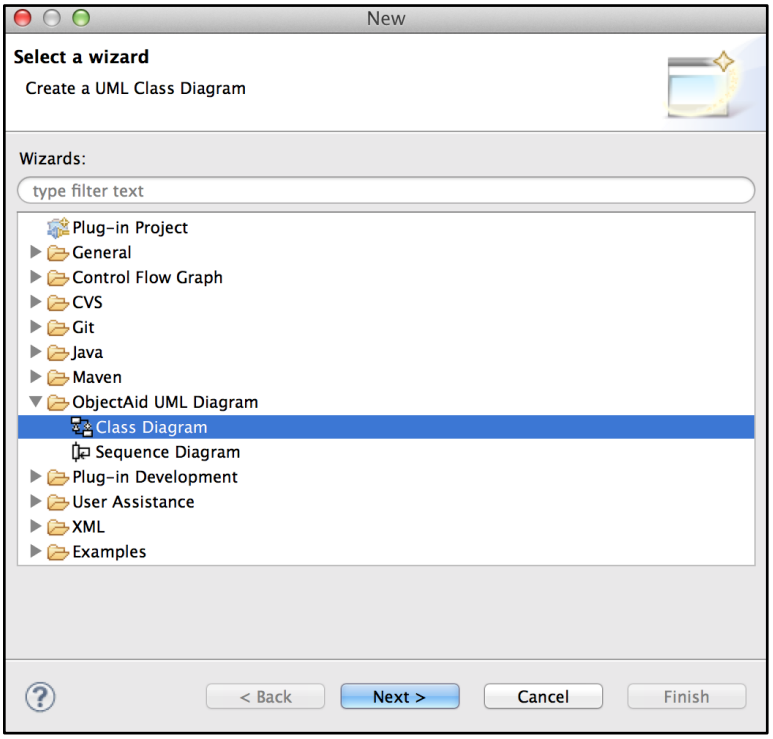
5.1.2 Get UML Class Diagrams

- Let's use the package "org.apache.tika.parser.html" in the project "tika-parsers" as an example;

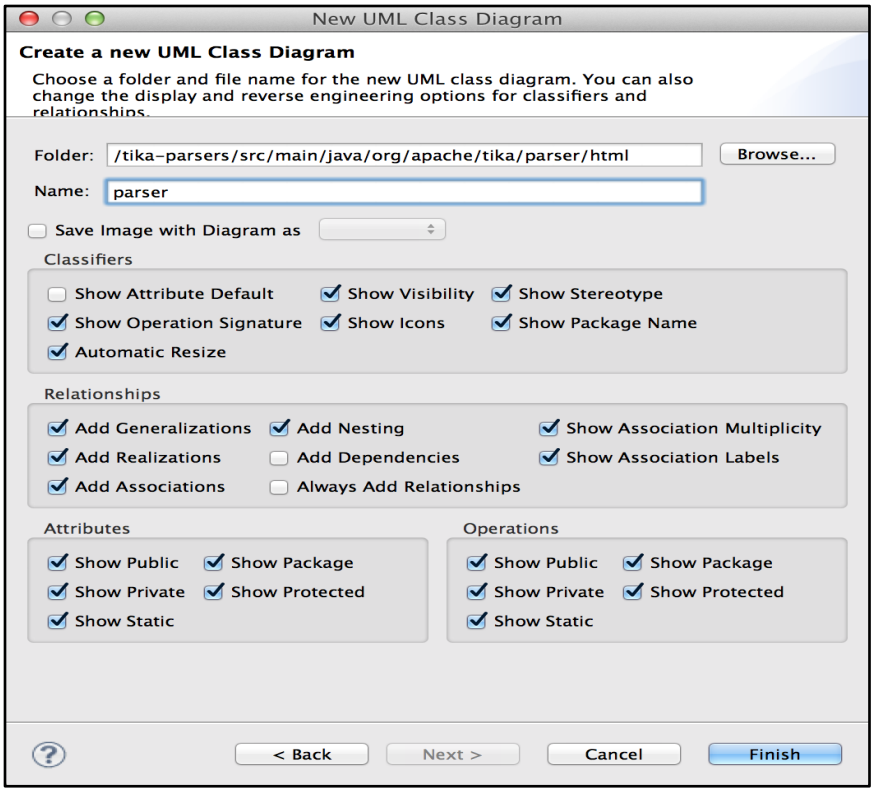
- Right click on the package, new -> other;



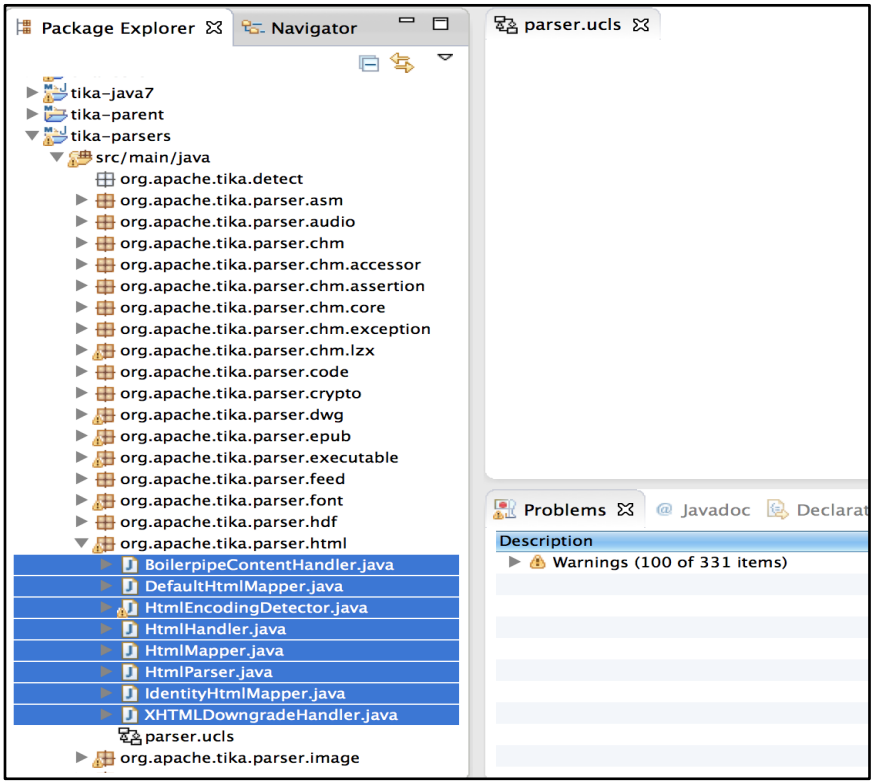
- Choose “Class Diagram”, press next;



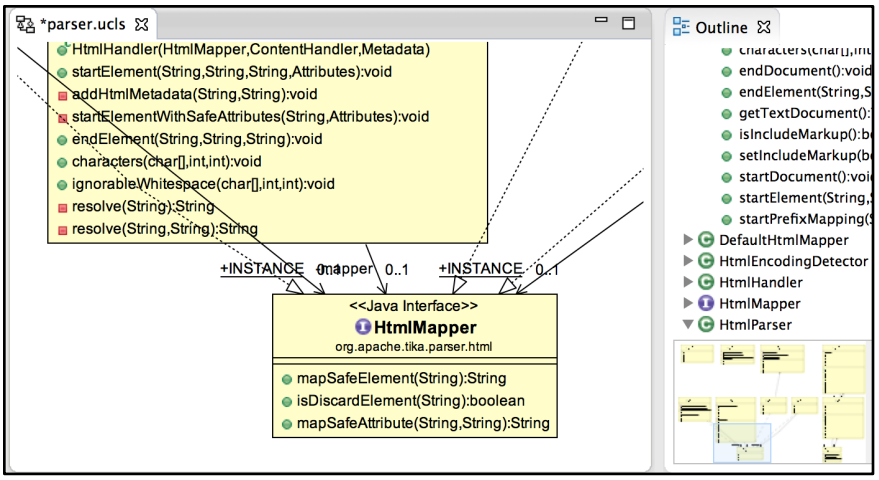
- Let’s use “parser” as the name of this class diagram, press finish;



- Select all the java files in that package, then drag them to the “parser.ucls”;

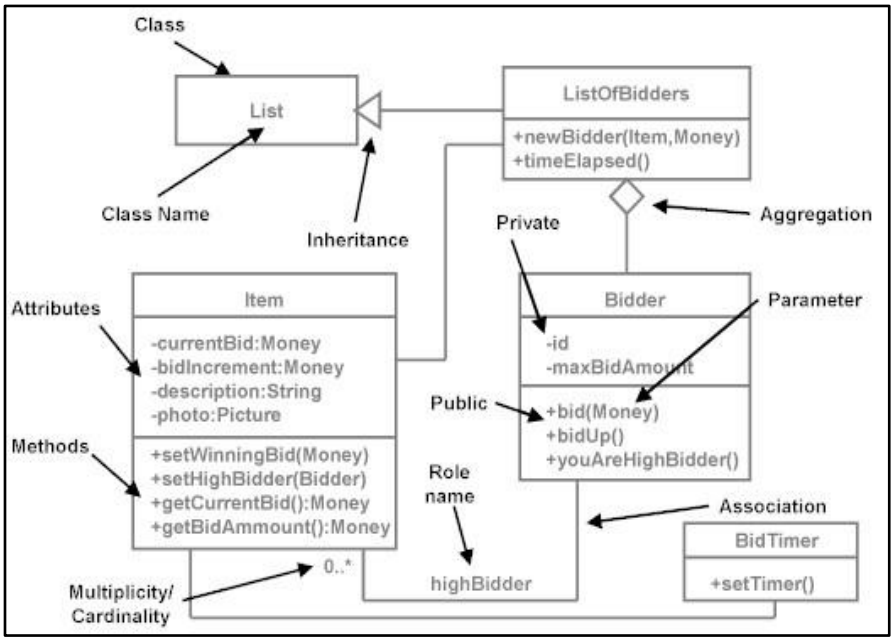


- Now we get an UML Class Diagram of that package;



5.1.3 Why it would be helpful for testing?

A class diagram in the UML is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.



With the help of class diagram, we can quickly get those information of a lot of classes. Then we can verify whether a class is complete, correct, consistent by doing syntax testing. A class diagram has abstracted those important information so that we don't need to understand a class by reading the whole file line by line before doing testing.

What's more, the relationship of classes are also shown in the class diagram. Becauses of that, when doing testing, we can know how to design test cases to satisfy the relationship of classes.

5.1.4 Other UML Class Diagrams

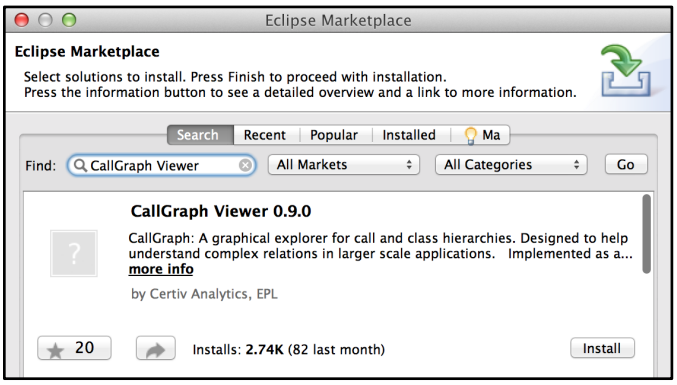
Please check the Reverse Engineering folder.

5.2 Call Graph

5.2.1 Installation;

- On the Eclipse menu bar, select Help -> Install New Software;

- Search “CallGraph”, press install;

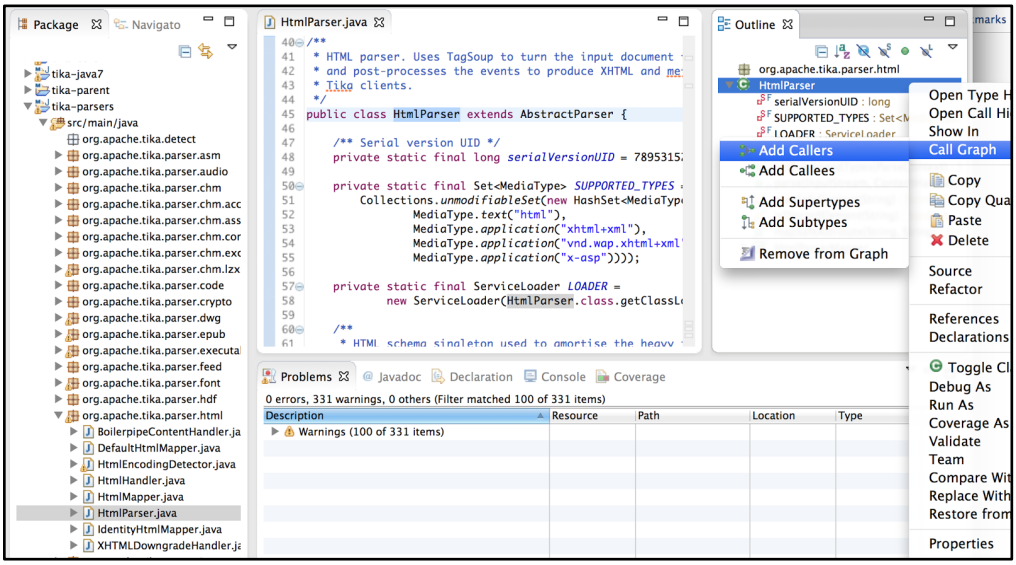


- Check box, follow the steps in the wizard to finish installation;

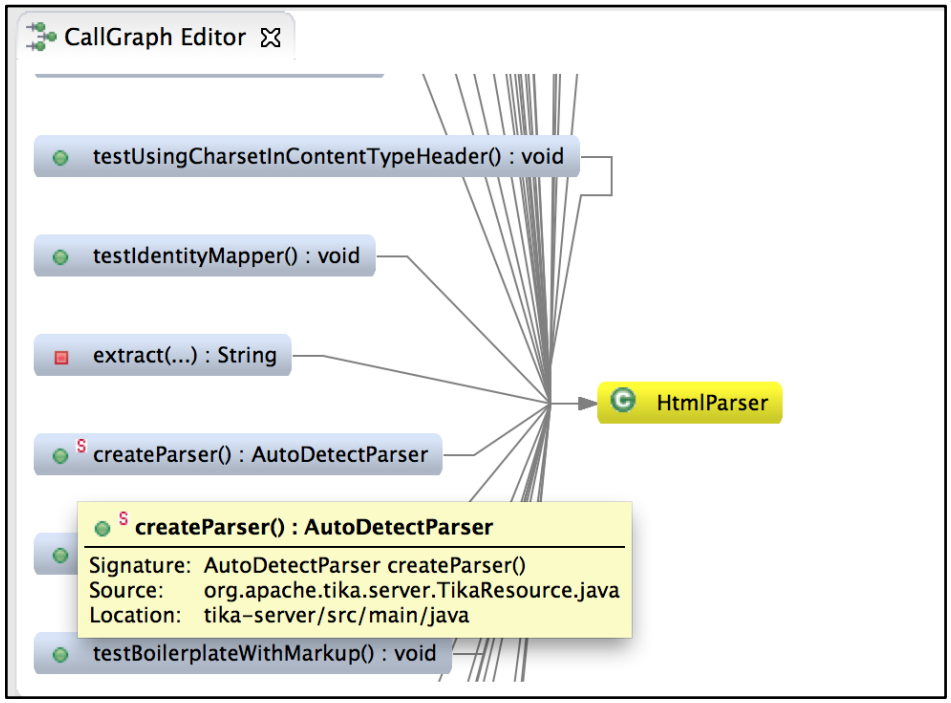
5.2.2 Get Call Graph

- Let’s use the java file “HtmlParser.java” in the the package “org.apache.tika.parser.html” in the project “tika-parsers” as an example;

- Open that java file, right click on the class name in Outline window. Call Graph -> Add Callers;



- Now, we get the call graph of the class HtmlParser. We can even check all the caller's information and quick open that caller file by double click on the caller name; Besides, if we want to know all the callers of the class, we can just choose "Add Callers" in the Call Graph;



5.2.3 Why it would be helpful for testing?

A call graph (also known as a call multigraph) is a directed graph that represents calling relationships between subroutines in a computer program. Specifically, each node represents a procedure and each edge (f, g) indicates that procedure f calls procedure g.

Call graphs are a basic program analysis result that can be used for human understanding of programs. When doing testing, call graphs can help us track the flow of values between procedures. Also, if we have changed a class, we know what classes may be probably influenced. Thus we can design test cases especially for those classes to make sure the consistency of the whole program.

5.2.4 Other Call Graphs

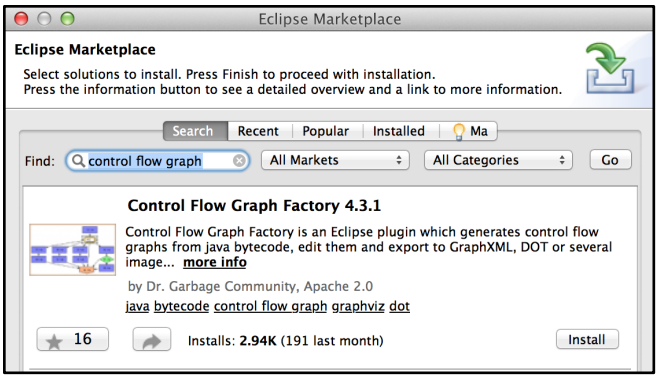
Please check the Reverse Engineering folder.

5.3 Control Flow Graphs

5.3.1 Installation;

- On the Eclipse menu bar, select Help -> Install New Software;

- Search "Control Flow Graph Factory", press install;

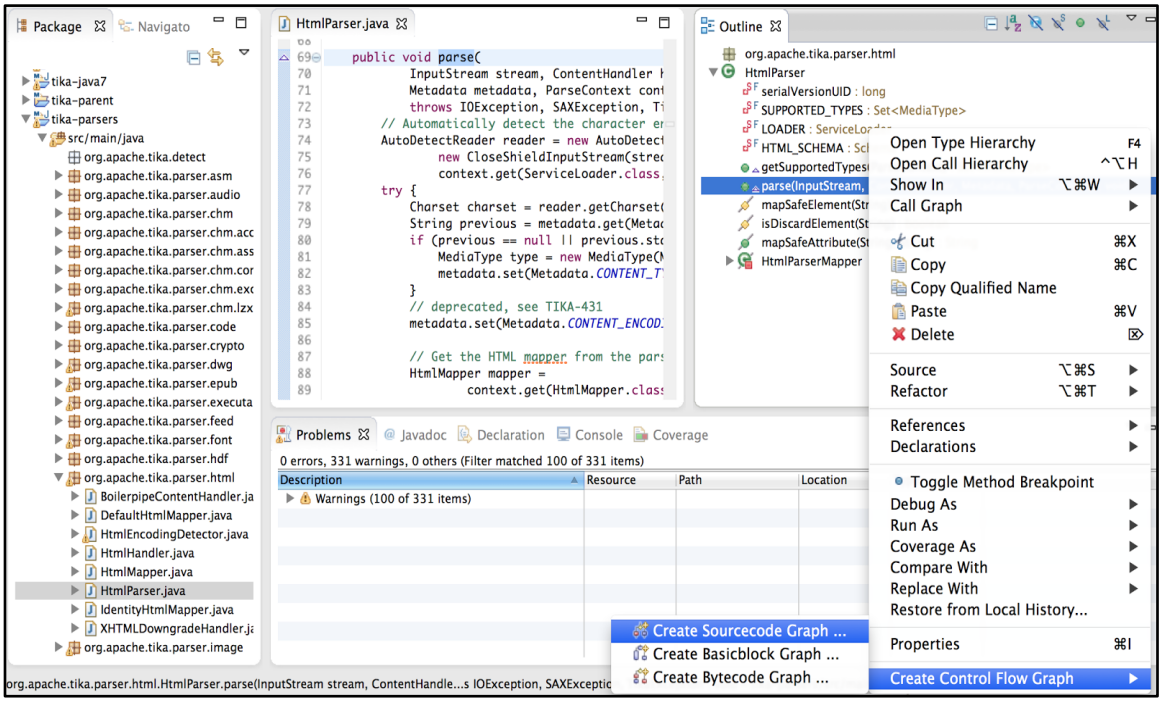


- Check box, follow the steps in the wizard to finish installation;

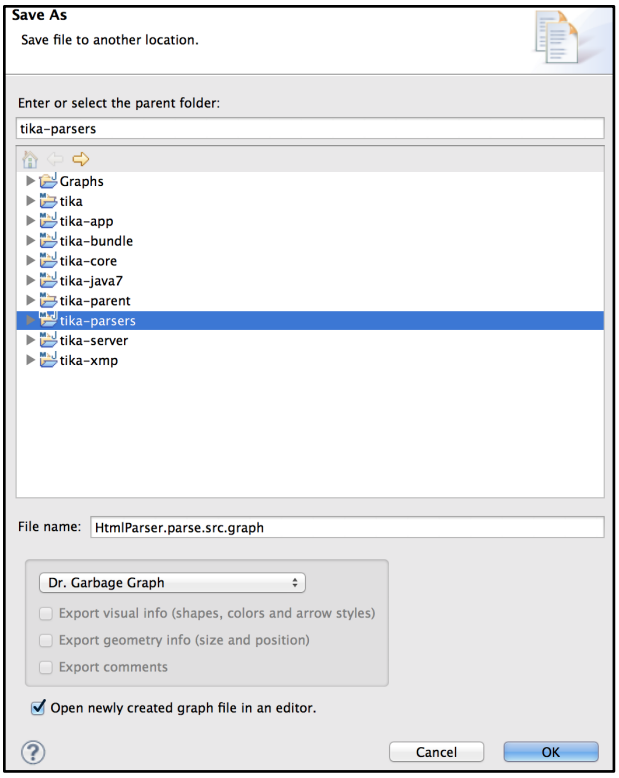
5.3.2 Get Control Flow Graphs

- Let's use the method "parse(InputStream, ContentHandler, Metadata, ParseContext)" in the java file "HtmlParser.java" in the the package "org.apache.tika.parser.html" in the project "tika-parsers" as an example;

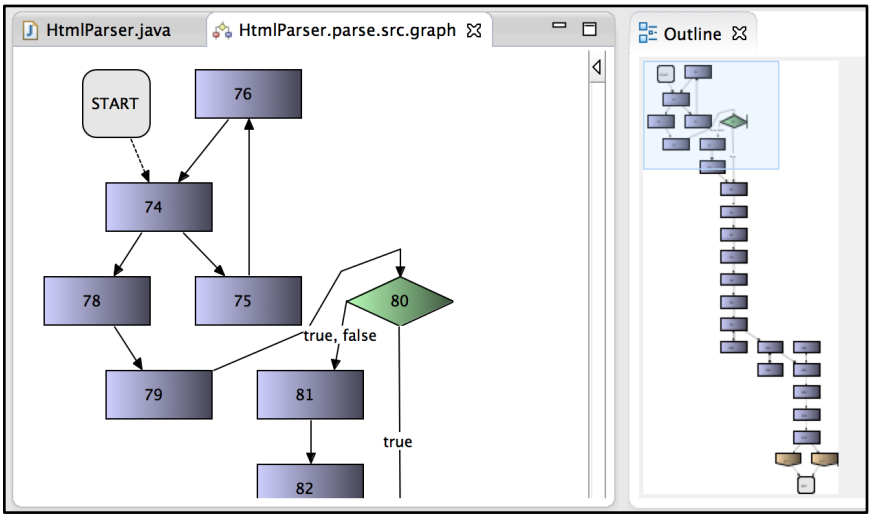
- Right click on the method in Outline window, Create Control Flow Graph -> Create Source code Graph;



- Save it in project "tika-parsers", press OK;



- Now we get the Control Flow Graph of that method. The number of a node means the line number of a statement;



5.3.3 Why it would be helpful for testing?

A control flow graph (CFG) is a representation of a program, using graph notation. It's a directed graph. Each node in that graph represents a statement or basic blocks. Each edge in that graph represents control flow (i.e. what happens after what).

By referring the control flow graph, we can design structural testing cases to coverage

statements or coverage branches as much as possible. We can know what is missing in our test suite. CFG is useful in optimizing our test cases.

5.3.4 Other Control Flow Graphs

Please check the Reverse Engineering folder.

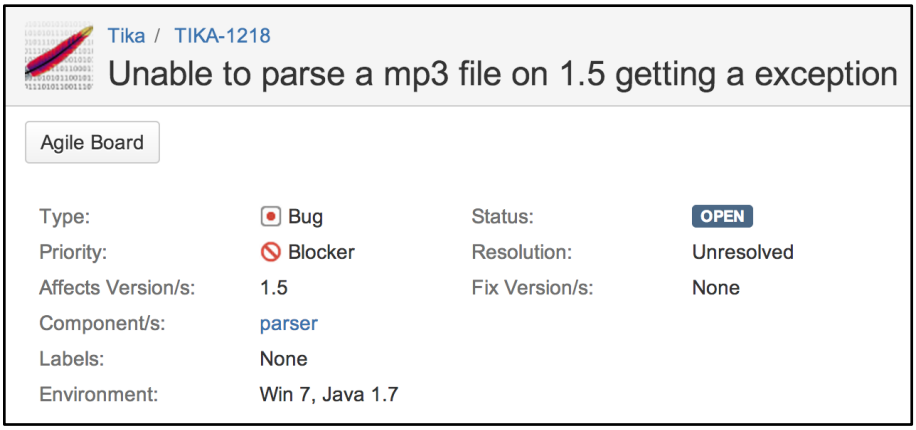
6. Debugging

Find a documented and verified bug report that has no automated test case that reproduces it, and create such an automated test case.

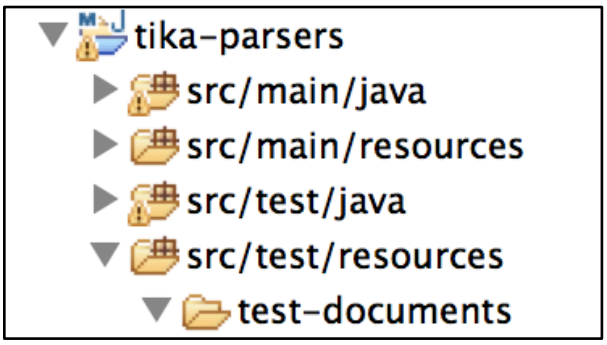
6.1 Unable to parse a mp3 file on 1.5 getting a exception.

(check here: https://issues.apache.org/jira/browse/TIKA-1218, for more details)

It is said that Tika is unable to parse a Mp3 file with a size bigger than 335387 bytes. Therefore, we use a Mp3 file called "Save-the-World-Knife-Party-Remix.mp3", which is bigger than that size, from that website as the test file.



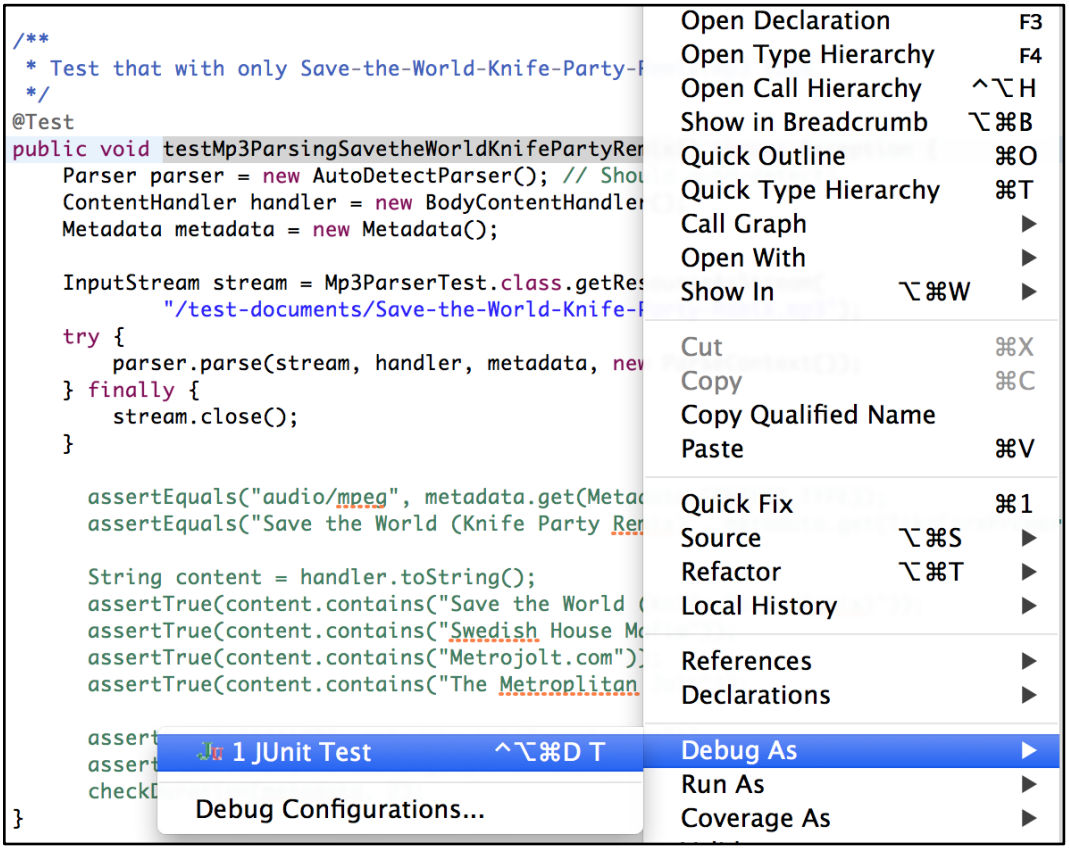
6.1.1 First, we download the Save-the-World-Knife-Party-Remix.mp3 file and put it into the test resources, just drag the file into the test-documents folder in the project "tika-parsers";



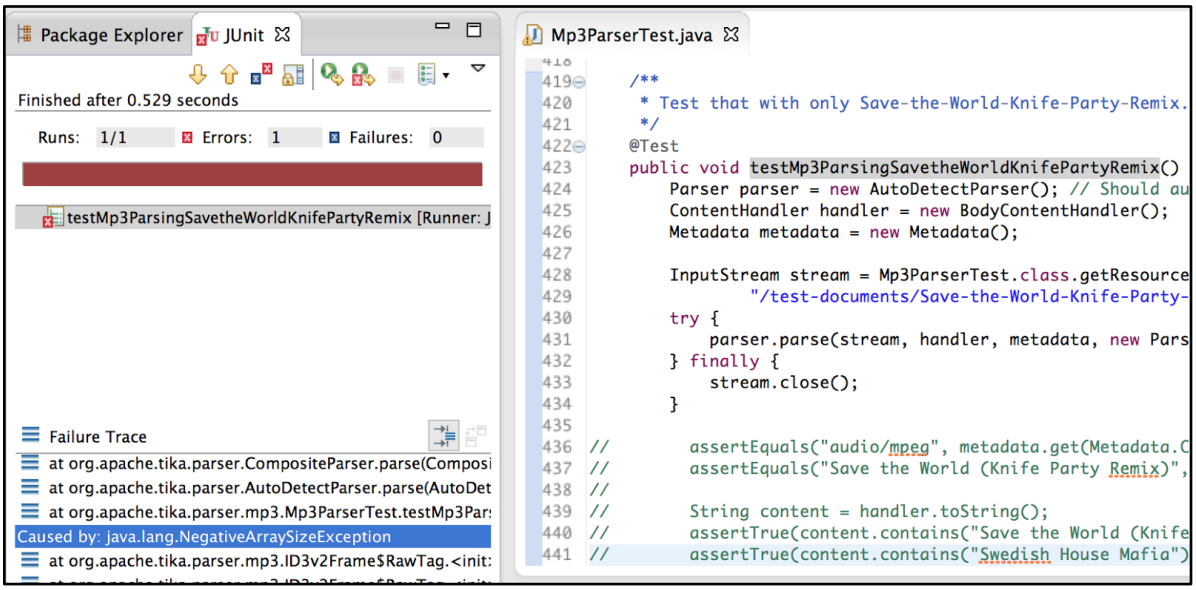
6.1.2 We create a new test case in the package "org.apache.tika.parser.mp3" in the folder "src/test/java" of the project "tika-parser"; Actually there is already an test file. What we need to

do is to add one more test case to test the reported bug; The test case is shown in the picture;

6.1.3 Right click on the method name, Debug As -> JUnit Test;

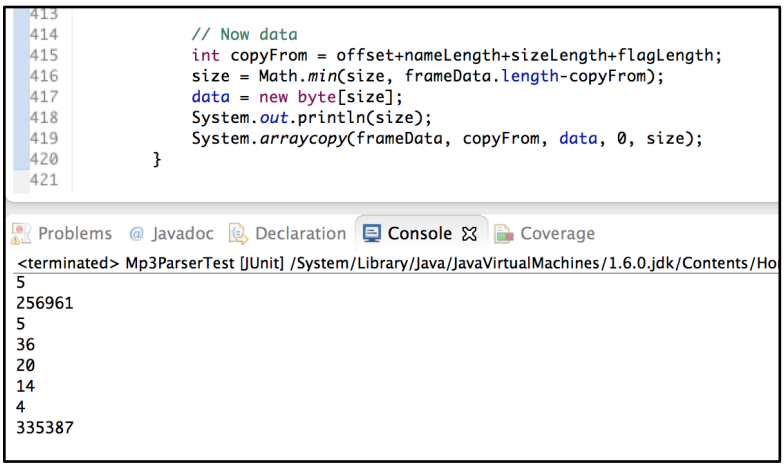


6.1.4 We can see that there is an error. Also, we can see check the Failure Trace window and got the information that it is caused by java.lang.NegativeArraySizeException;



6.1.5 If we click on the Failure Trace window information, we can go to the place the error happened. It's in the file "ID3v2Frame.java";

6.1.6 There is a commented out statement "System.out.println(size);" It's written by our team. If we active that statement and do the Junit test again, we can see the last result in the console is 335387 as shown in the next picture. That means before the error occurs, the max size is 335387 bytes, while our test Mp3 file is definitely bigger that size. Hence, there comes the error.



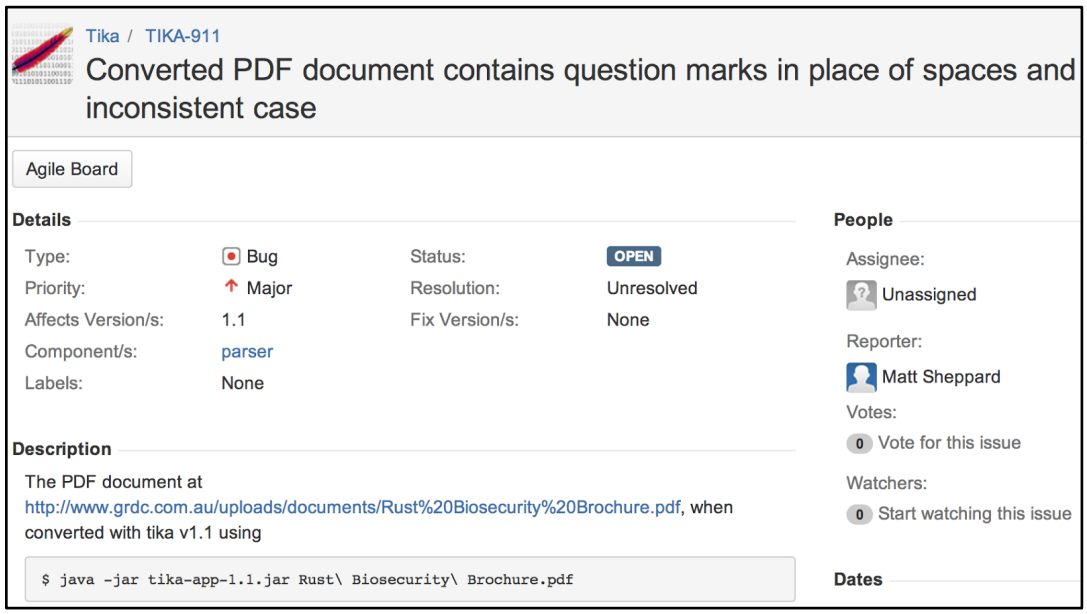
6.1.7 Discussion

This is a bug that need to be solved. Since the size of byte array is too limited. Maybe in the future Tika, it's better to use ArrayList to dynamically store some data.

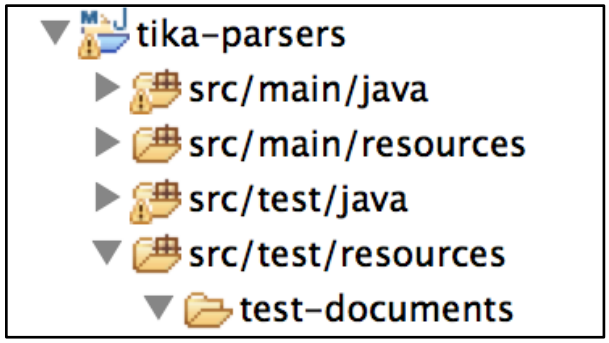
6.2. Converted PDF document contains question marks

(check here: https://issues.apache.org/jira/browse/TIKA-911, for more details)

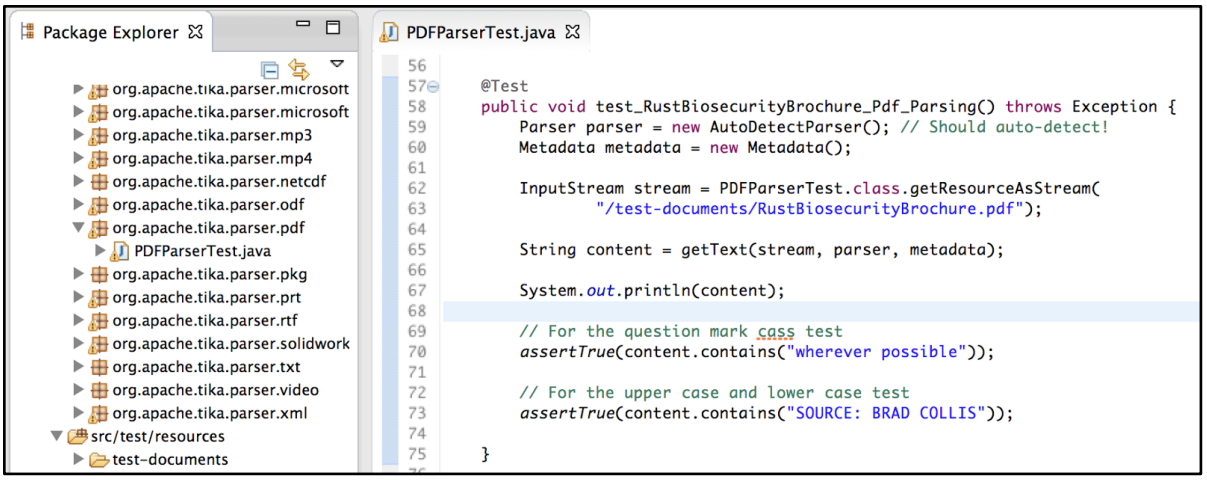
It is said that when parsing PDF files with Tika, some “spaces” will be replaced with question marks. Besides, the upper and lower case of some words will be changed.



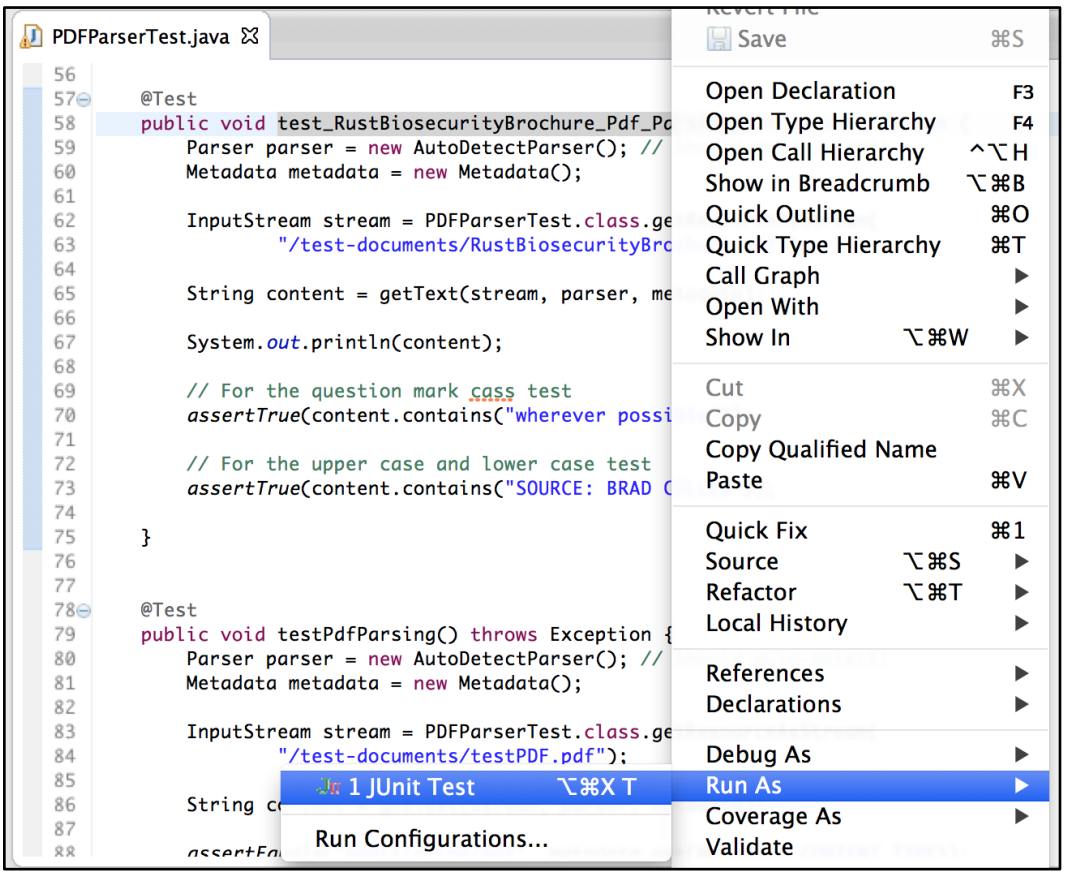
6.2.1 First, we download the RustBiosecurityBrochure.pdf file from that website and put it into the test resources, just drag the file into the test-documents folder in the project “tika-parsers”;



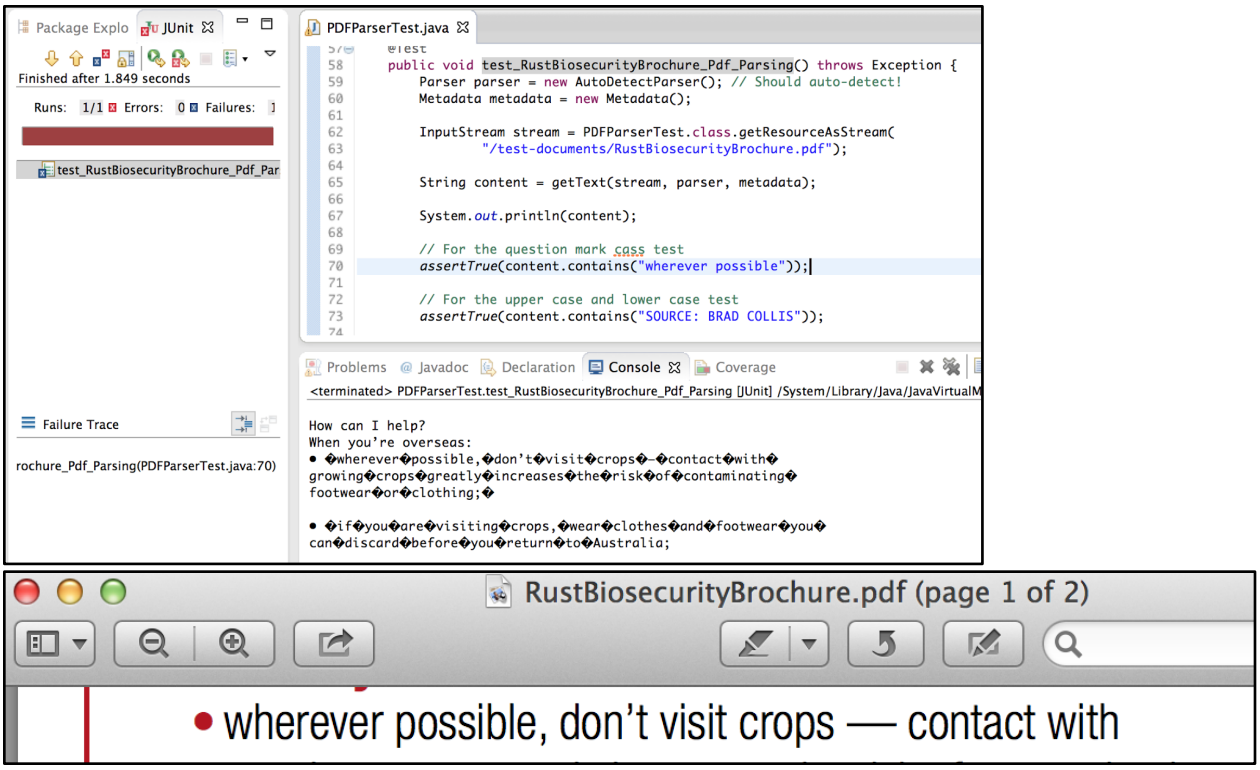
6.2.2 We create a new test case in the package “org.apache.tika.parser.pdf” in the folder “src/test/java” of the project “tika-parser”; Actually there is already an test file. What we need to do is to add one more test case to test the reported bug; The test case is shown in the picture;



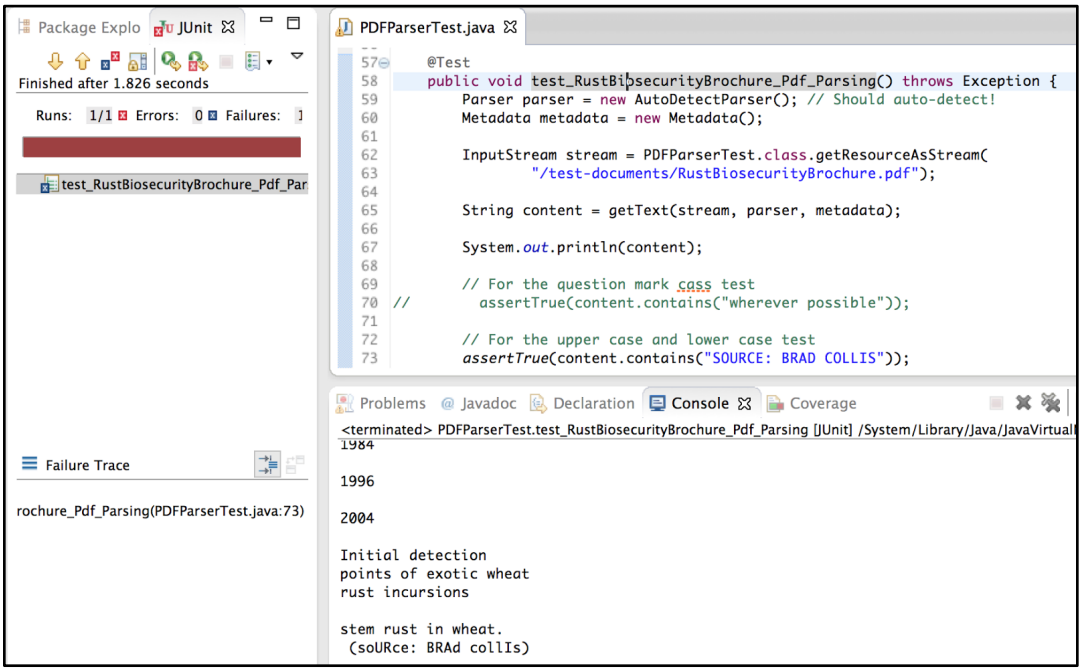
6.2.3 Right click on the method name, Debug As -> JUnit Test;

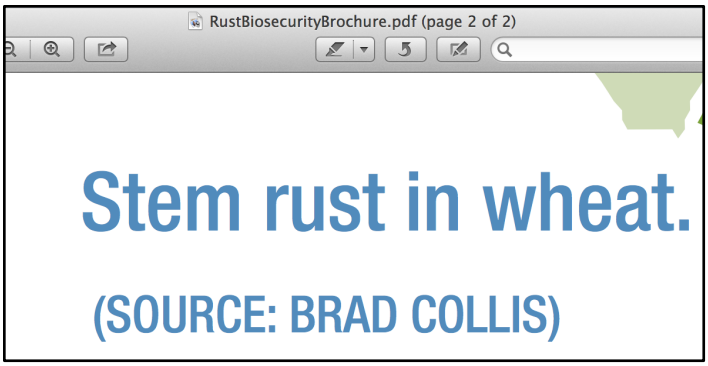


6.2.4 We can see it failed; If we check the Failure Trace window, we can know that it failed because of the statement of line 70. For quick compare, I also have printed out the parsed content in the console window. The original content contains “wherever possible”, however, after parsing, the result shows that there comes a question mark between these two words. It Failed. That’s a bug;



6.2.5 If we commented out line 70 and do the test again. This time, it failed because the words "SOURCE: BRAD COLLIS" in the original file were parsed to "soURce: BRAd collIs". The bug is about inconsistent case;





6.2.6 Discussion

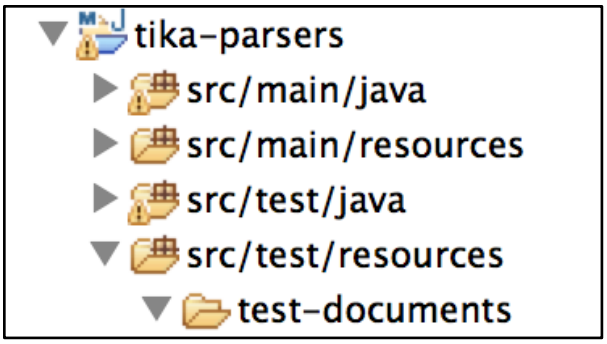
Those two small bugs need to be solved. Before releasing next version of Tika, the testers need to test those two bugs first.

6.3 Embedded object not extracted.

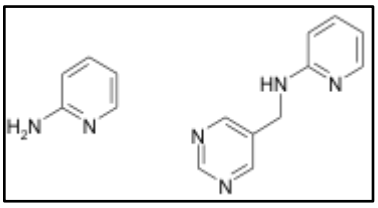
(check here: https://issues.apache.org/jira/browse/TIKA-1167, for more details)

Given embedded element (image of wmf format in a document in docx) It is seems that Tika could detect the embedded object. The element could be tagged as , however extraction itself (using -z on the command line, or using API) does not seem to work for this object.

6.3.1 First, we download the testing document Doc_w_Structre_that_wont_extract.docx and put it into the test resources, by dragging the file into the test-documents folder in the project "tika-parsers";



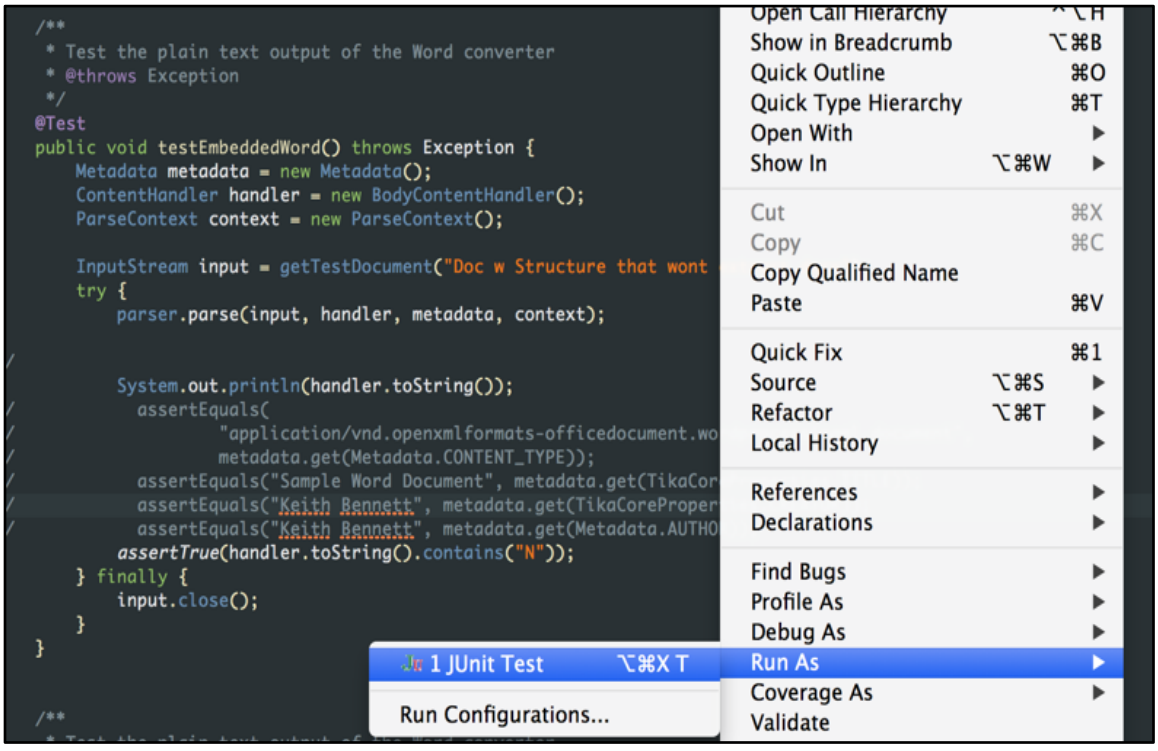
the embedded image in this docx is as follows



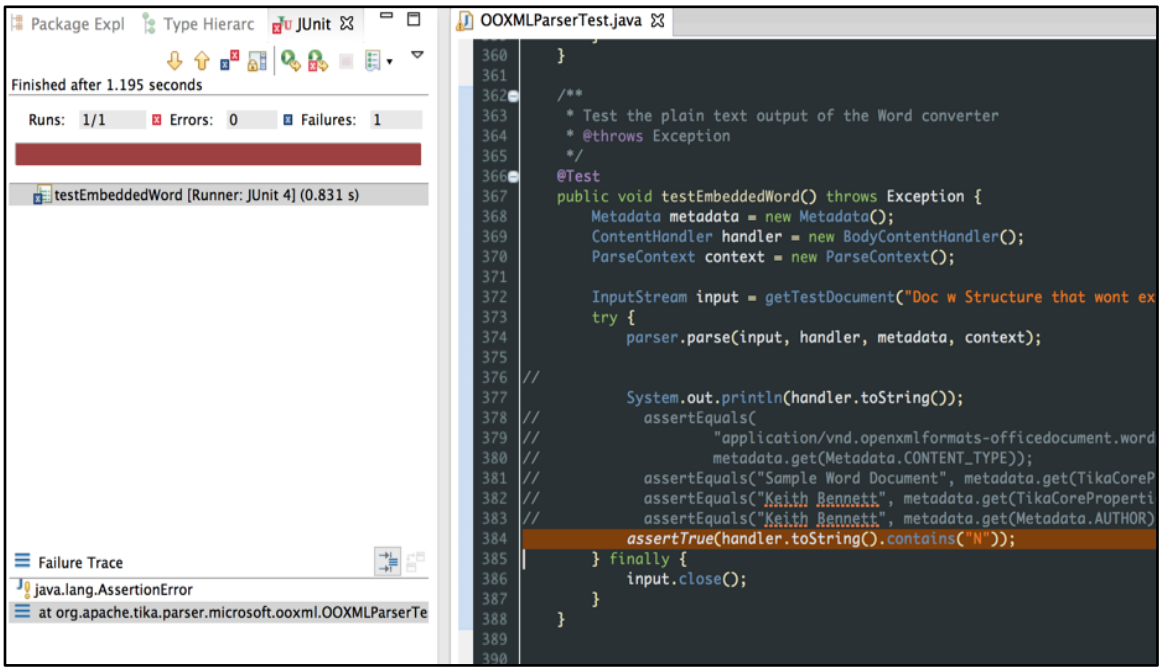
tika should extract the text “H2N” “N” “HN” from it.

6.3.2 We create a new test case in the package “org.apache.tika.parser.microsoft.ooxml” in the folder “src/test/java” of the project “tika-parser”; Actually there is already an test file. What we need to do is to add one more test case to test the reported bug; The test case is shown in the picture;

6.3.3 Right click on the method name, Debug As -> JUnit Test;



6.3.4 We can see this test method failed. By checking the status of the Failure Trace window, we can know that it failed because of the statement of line 384.



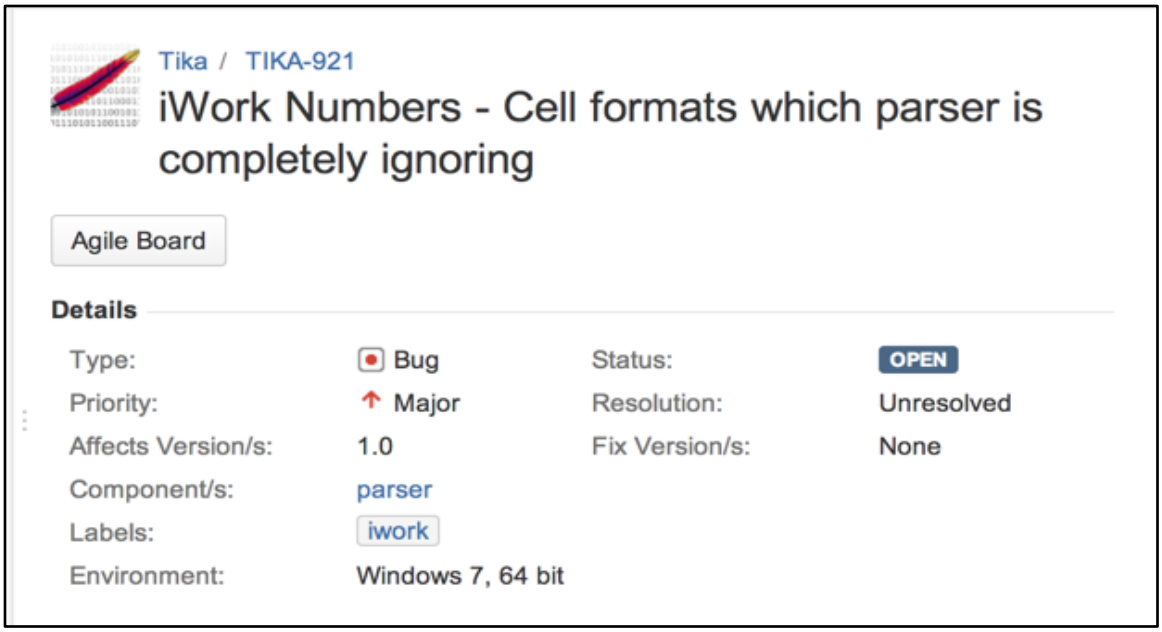
6.3.5 Discussion

For a quick review, I put a System.out.println() statement in the test method to print out the result in the console during the processing. However, in this case, the console is blank after JUnit running of this test method. The expected parse results, as described in previous, are “H2N” “HN” “N”. So there are possible two reasons that might cause this problem. One is that tika has successfully extract the embedded element from the document but fail to parse it, another one is that tika fails to extract the image element from the document. This bug should be resolved in the next generation of Tika. And developers need to take this test case into consideration.

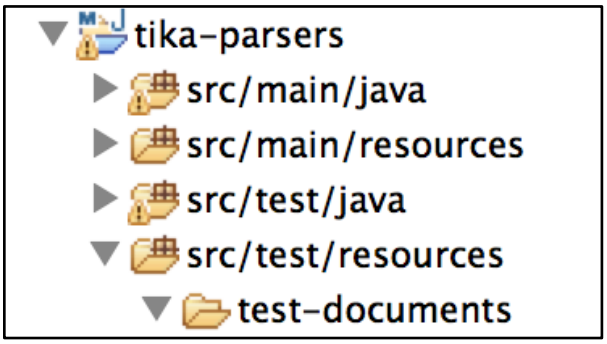
6.4 iWork Numbers - Cell formats which parser is completely ignoring

(check here: https://issues.apache.org/jira/browse/TIKA-921, for more details)

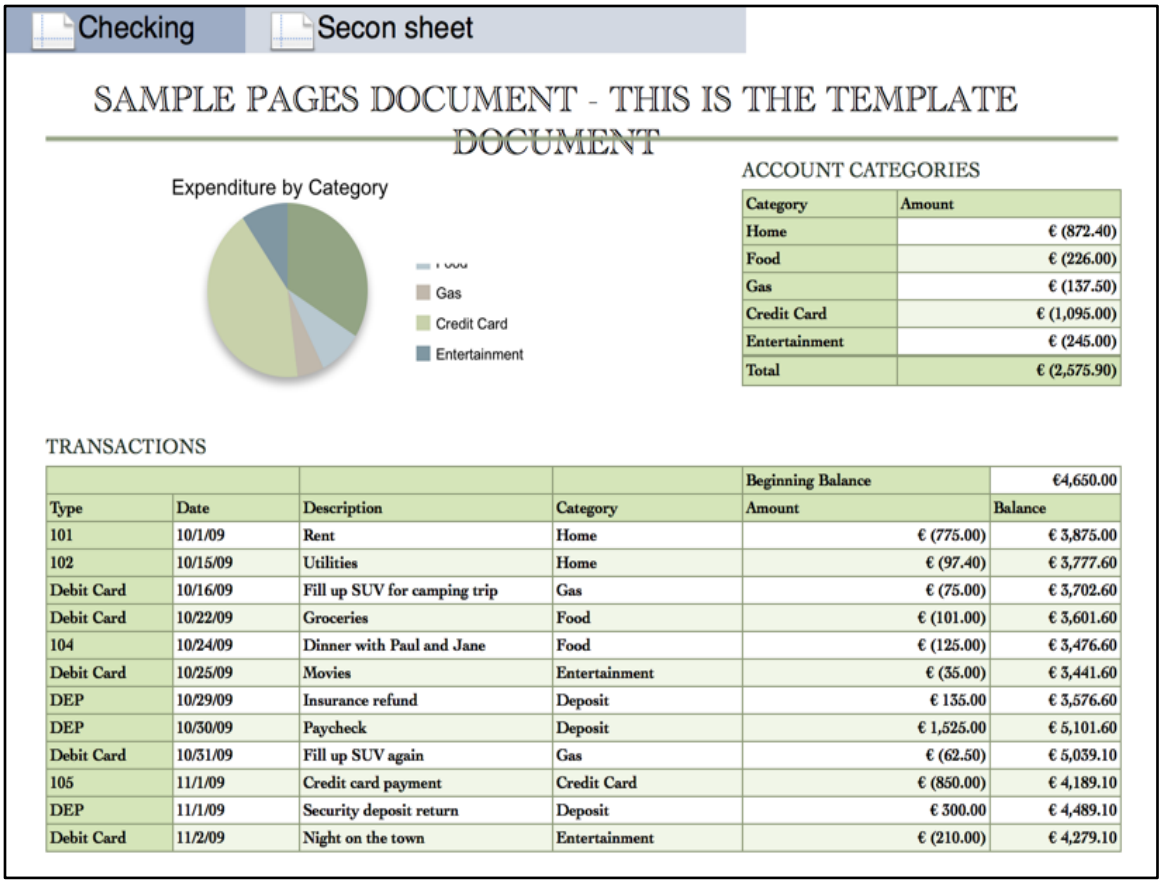
It seems that tika cannot extract the Date and Time, the length of the time, duration, stepper (incrementer) from the iWork Number cell formats.



6.4.1 First, we download the testing document testNumbersTemplate.numbers and put it into the test resources, by dragging the file into the test-documents folder in the project "tika-parsers";



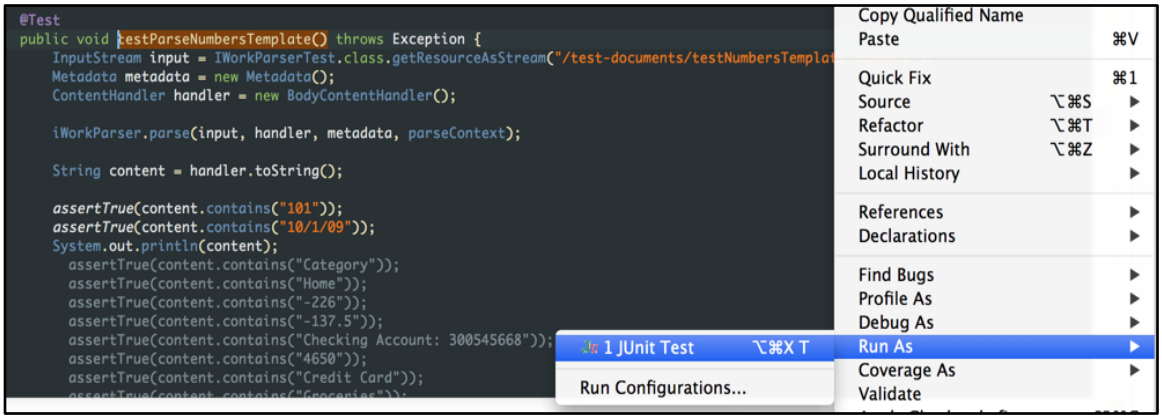
screenshot of the iWork Numbers document is as follows:



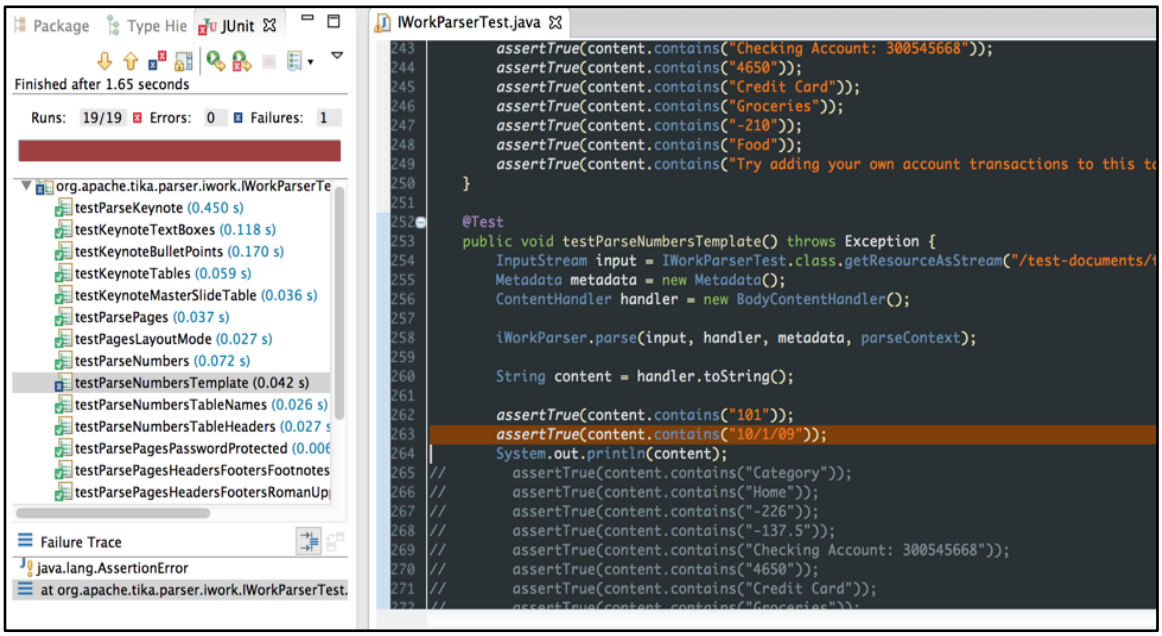
as we can see, this document has a variety of data, including data in date format.

6.4.2 We create a new test case in the package “org.apache.tika.parser.iwork” in the folder “src/test/java” of the project “tika-parser”; Actually there is already an test file. What we need to do is to add one more test case to test the reported bug; The test case is shown in the picture;

6.4.3 Right click on the method name, Debug As -> JUnit Test;

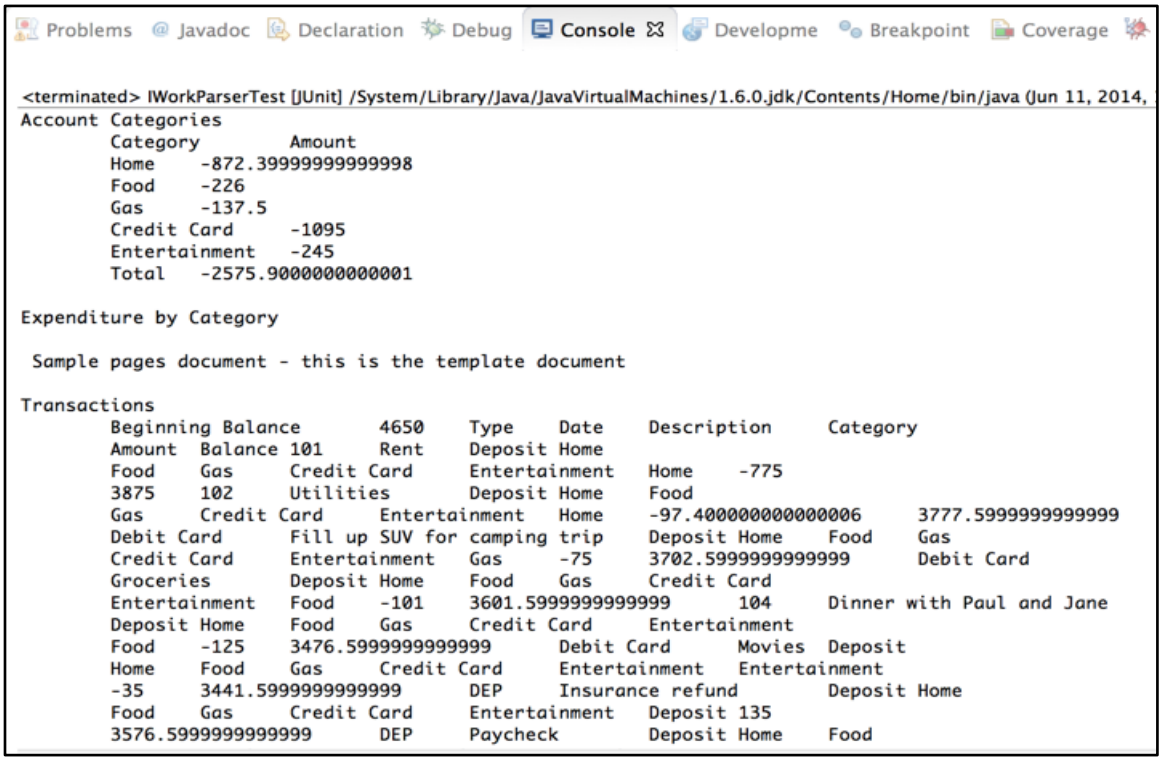


6.4.4 We can see this test method failed. By checking the status of the Failure Trace window, we can know that it failed because of the statement of line 263.



6.4.5 Discussion

For a quick review, I put a System.out.println() statement in the test method to print out the result in the console during the processing.

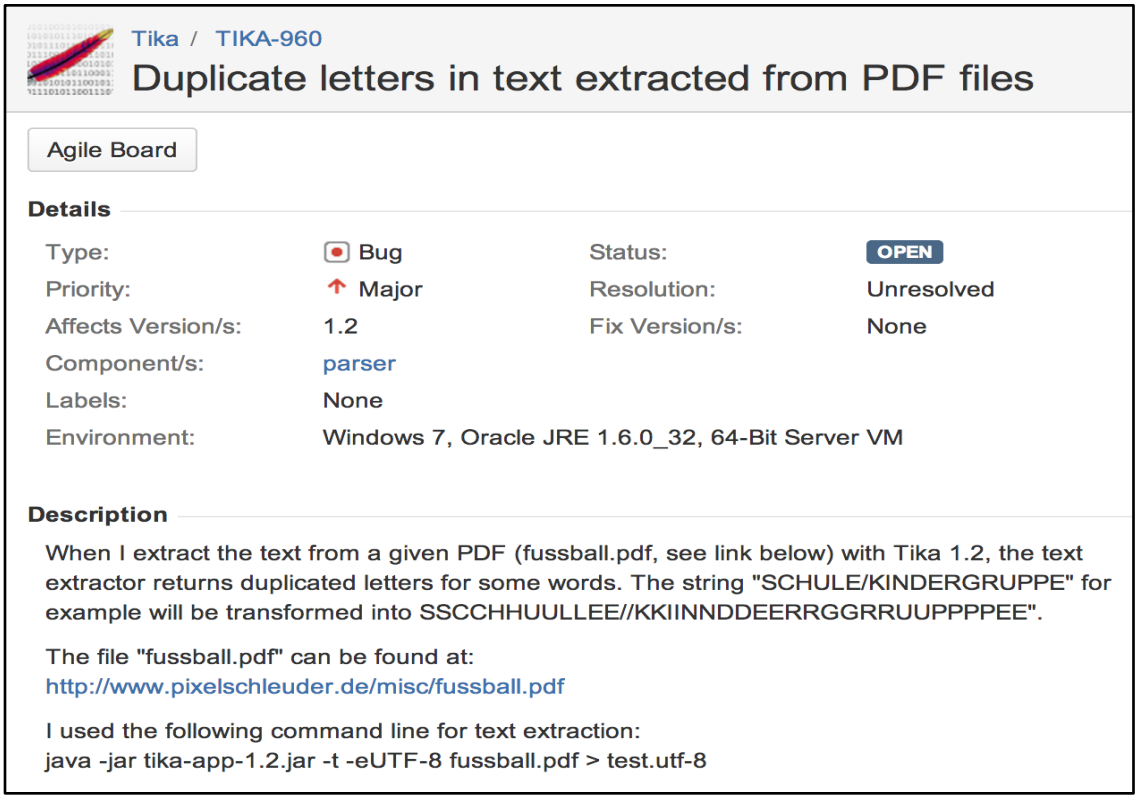


in the console window, we can see that all the information of date is missing. Also, there is some incorrect parse result. Like € (872.40) is parsed into -872.399999999998. This bug should be resolved in the next generation of Tika. And developers need to take this test case into consideration.

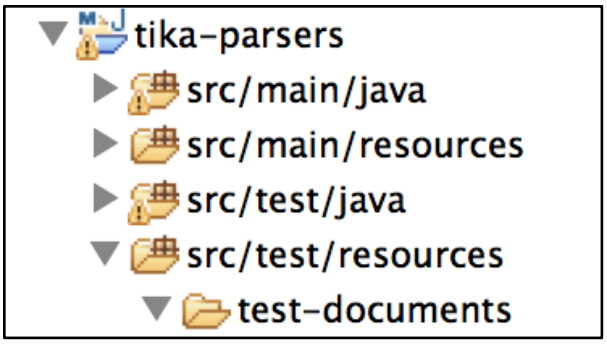
6.5 Duplicate letters in text extracted from PDF files

(check here: https://issues.apache.org/jira/browse/TIKA-960, for more details)

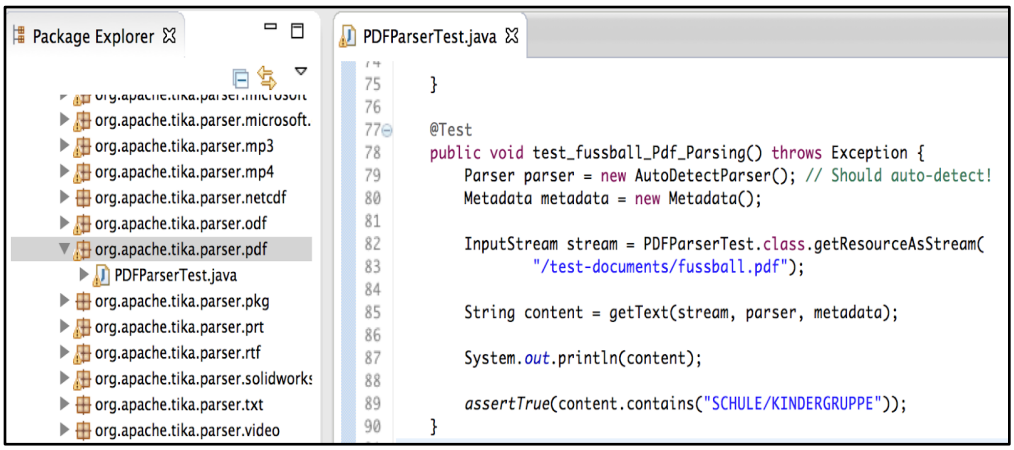
It is said that when parsing a file called “fussball.pdf” attached on that website with Tika, The string "SCHULE/KINDERGRUPPE" for example will be transformed into SSCCHHUULLEE//KKIINNDDEERRGGRRUUPPPPEE";



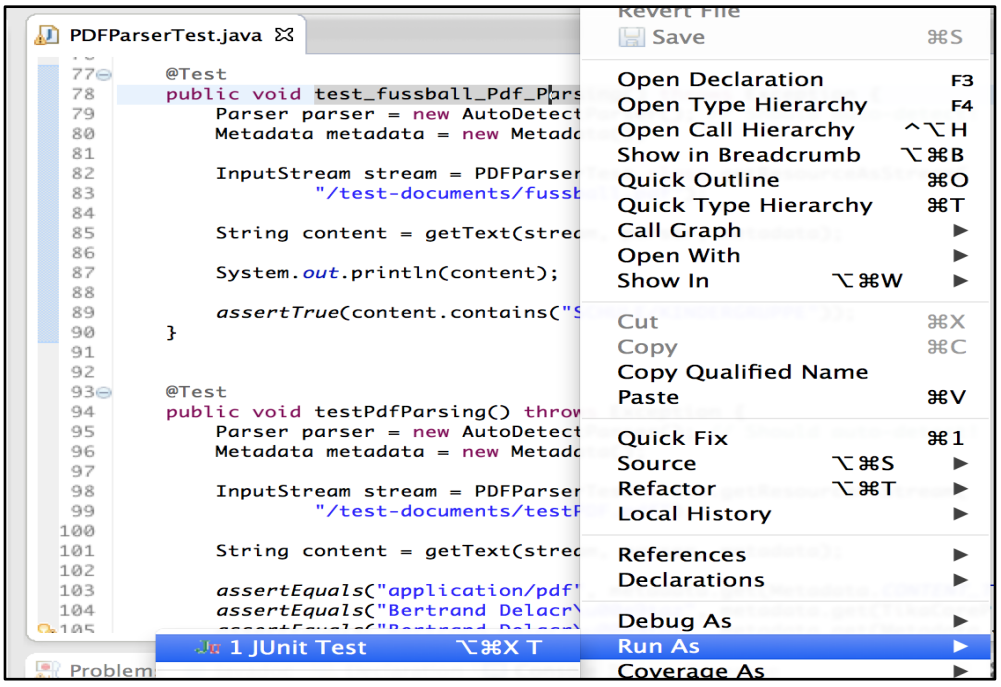
6.5.1 First, we download the fussball.pdf file from that website and put it into the test resources, just drag the file into the test-documents folder in the project “tika-parsers”;



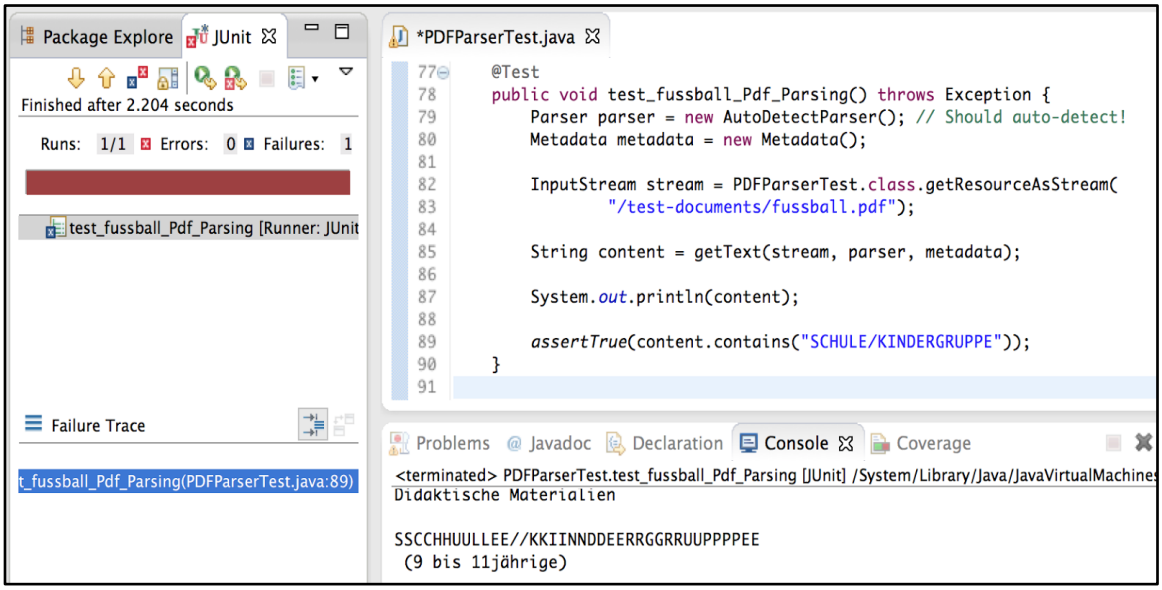
6.5.2 We create a new test case in the package “org.apache.tika.parser.pdf” in the folder “src/test/java” of the project “tika-parser”; Actually there is already an test file. What we need to do is to add one more test case to test the reported bug; The test case is shown in the picture;



6.5.3 Right click on the method name, Debug As -> JUnit Test;



6.5.4 We can see it failed; If we check the Failure Trace window, we can know that it failed because of the statement of line 89. For quick compare, I also have printed out the parsed content in the console window. The original content contains “SCHULE/KINDERGRUPPE”, however, after parsing, the result shows that there comes “SSCCHHUULLEE//KKIINNDDEERRGGRRUUPPPPEE”. That’s a bug;



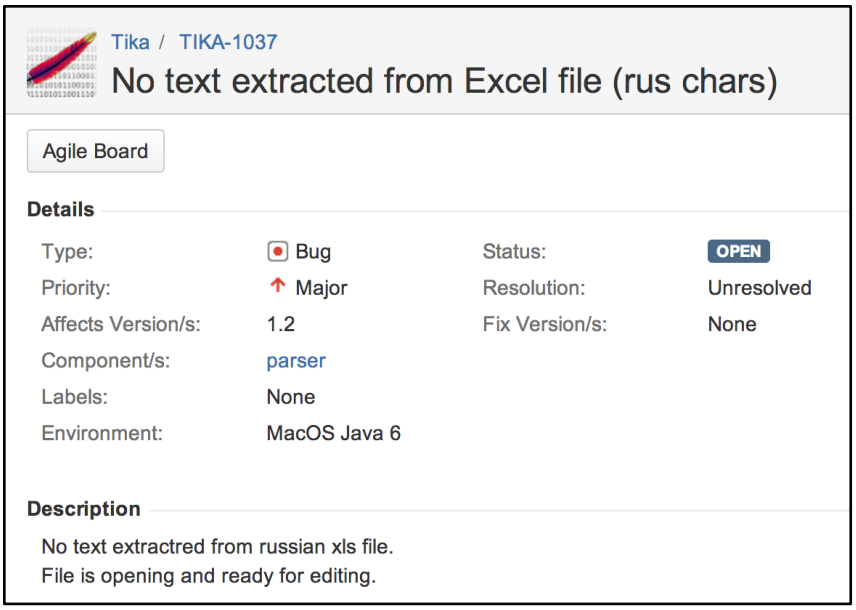
6.5.5 Discussion

I cannot get the possible reason causes this bug. Before releasing next version of Tika, the testers need to test this bug first.

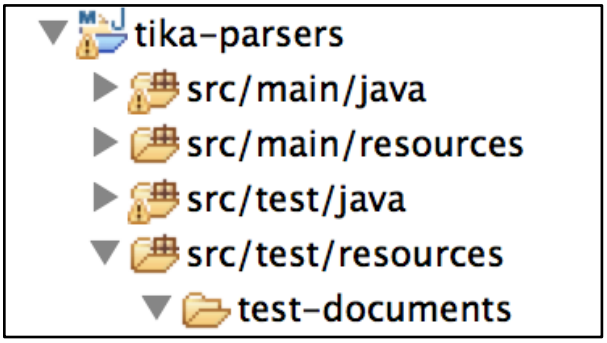
6.6 No text extracted from Excel file (rus chars)

(check here: https://issues.apache.org/jira/browse/TIKA-1037, for more details)

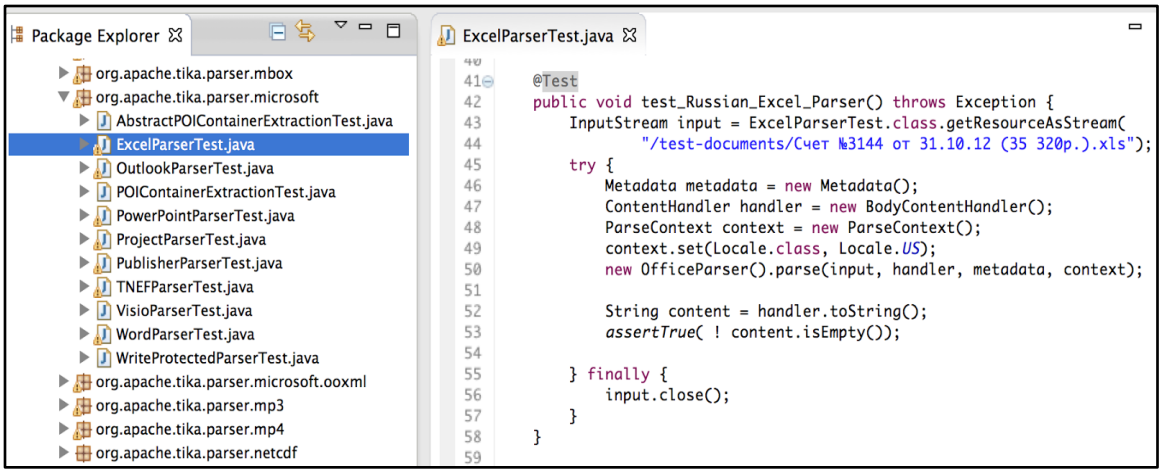
It is said that when Tika is parsing a Russian xls file, no text extracted



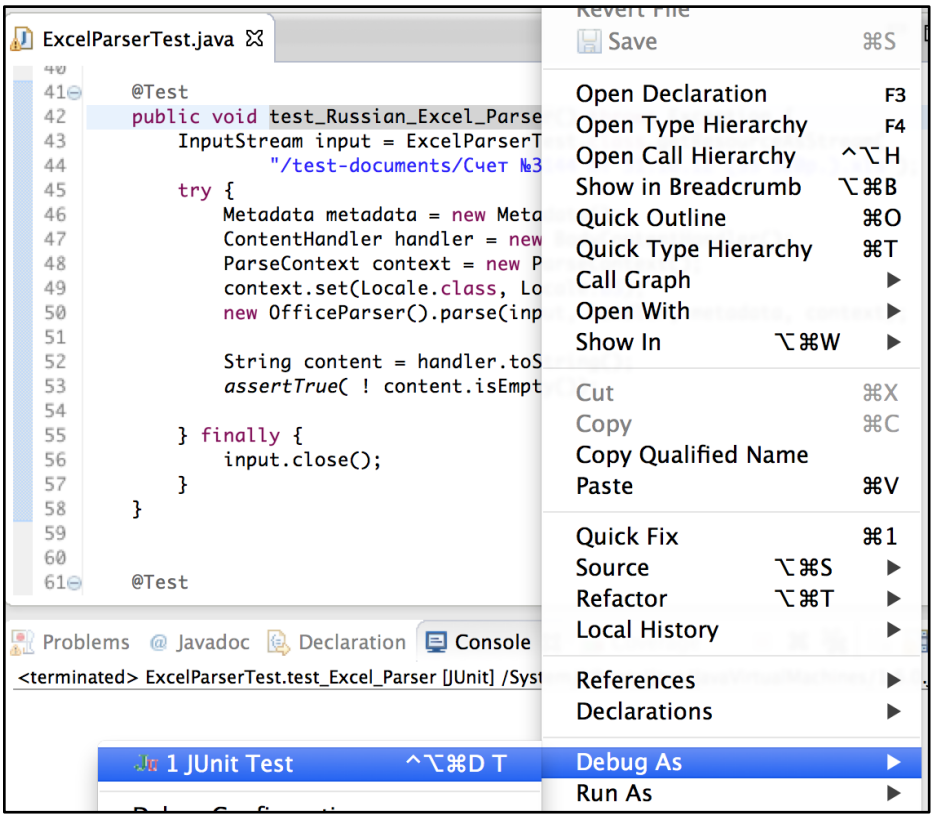
6.6.1 First, we download the “Счет №3144 от 31.10.12 (35 320р.).xls” file from that websites and put it into the test resources, just drag the file into the test-documents folder in the project “tika-parsers”;



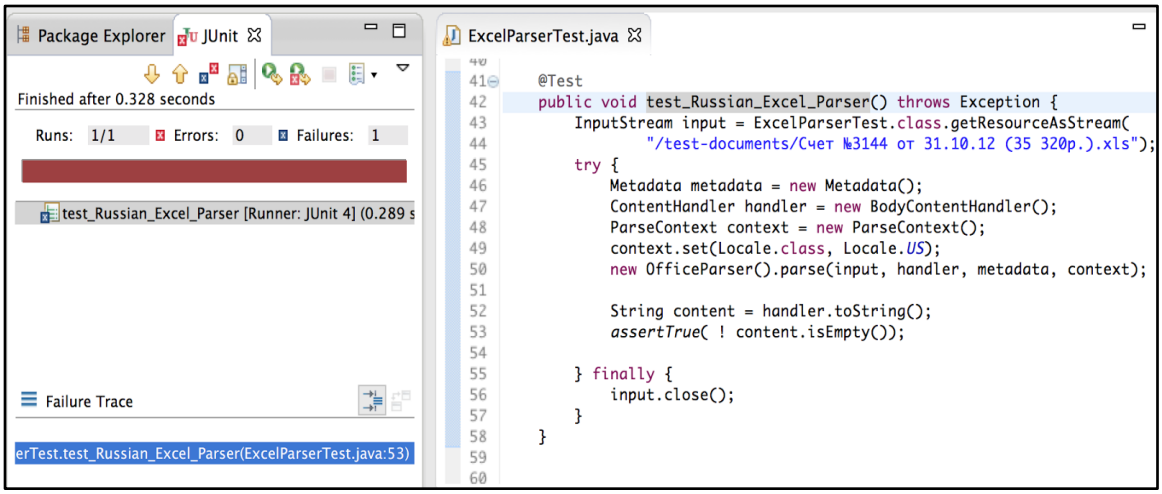
6.6.2 We create a new test case in the package “org.apache.tika.parser.microsoft” in the folder “src/test/java” of the project “tika-parser”; Actually there is already an test file. What we need to do is to add one more test case to test the reported bug in “ExcelParserTest.java”; The test case is shown in the picture;



6.6.3 Right click on the method name, Debug As -> JUnit Test;



6.6.4 We can see that there is an error. Also, we can see check the Failure Trace window and got the information that it is caused by statement on of line 53. That means the content is empty. However, the original file is not empty. That's a bug.



6.6.5 Discussion

This needs to be test before releasing next version of Tika.